

# Delays in Esterel

## Timothy Bourke

INRIA/IRISA

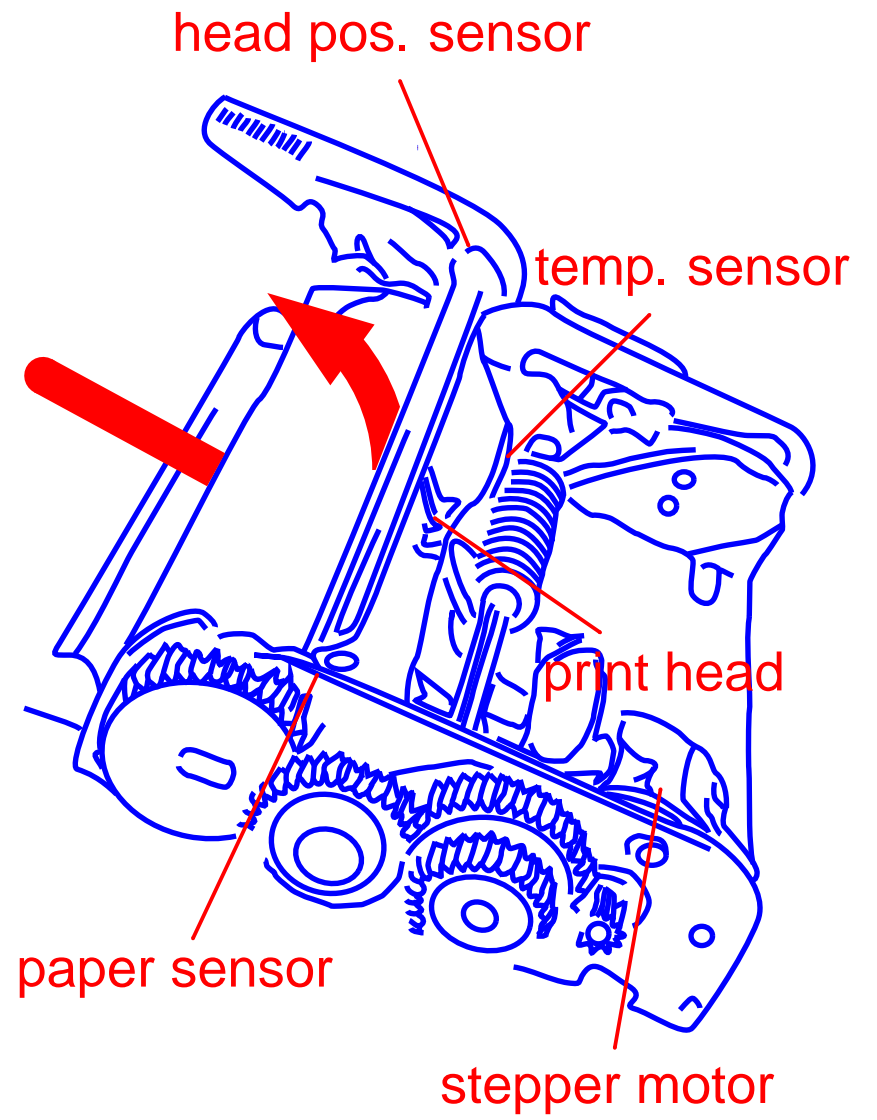
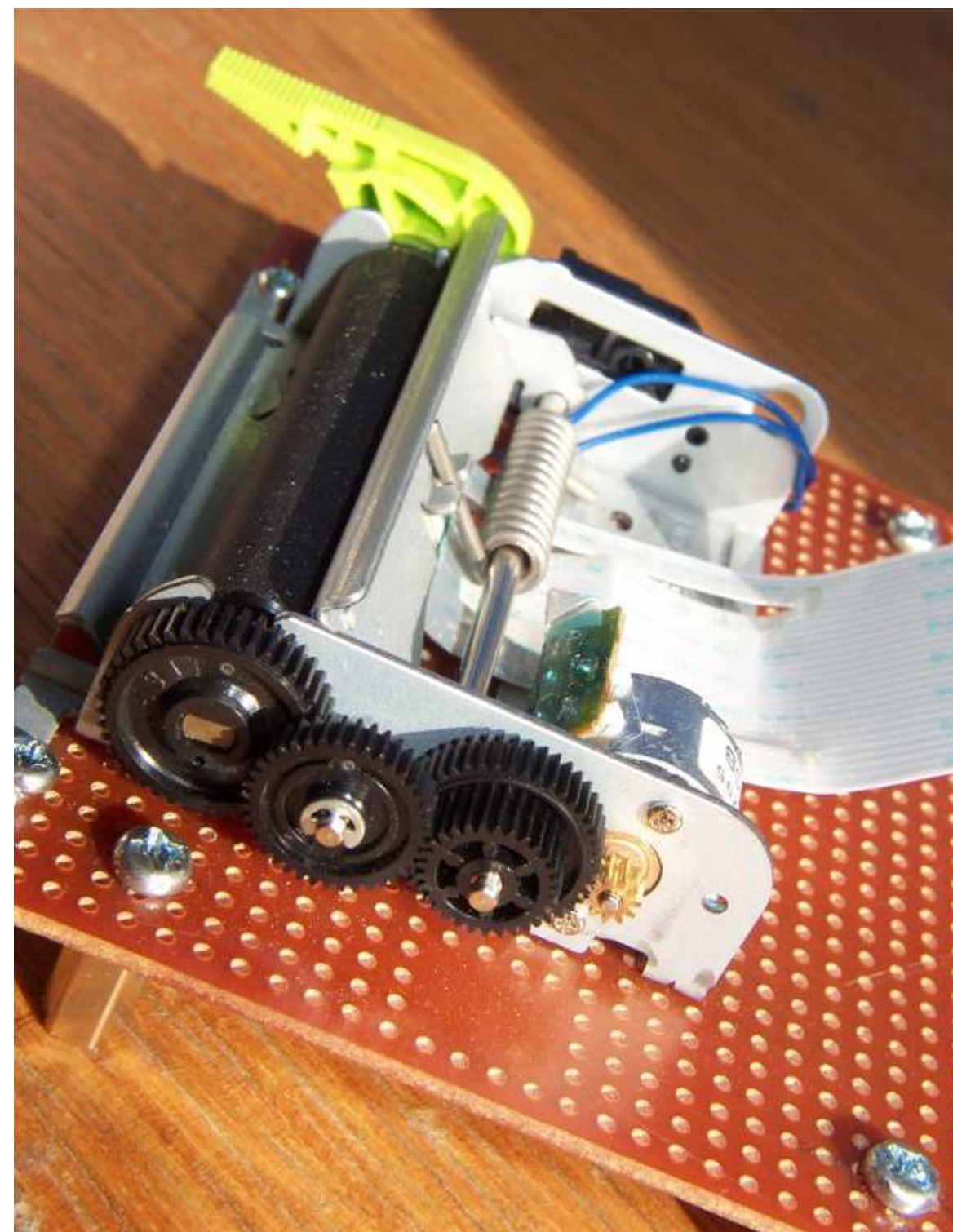
`timothy.bourke@irisa.fr`

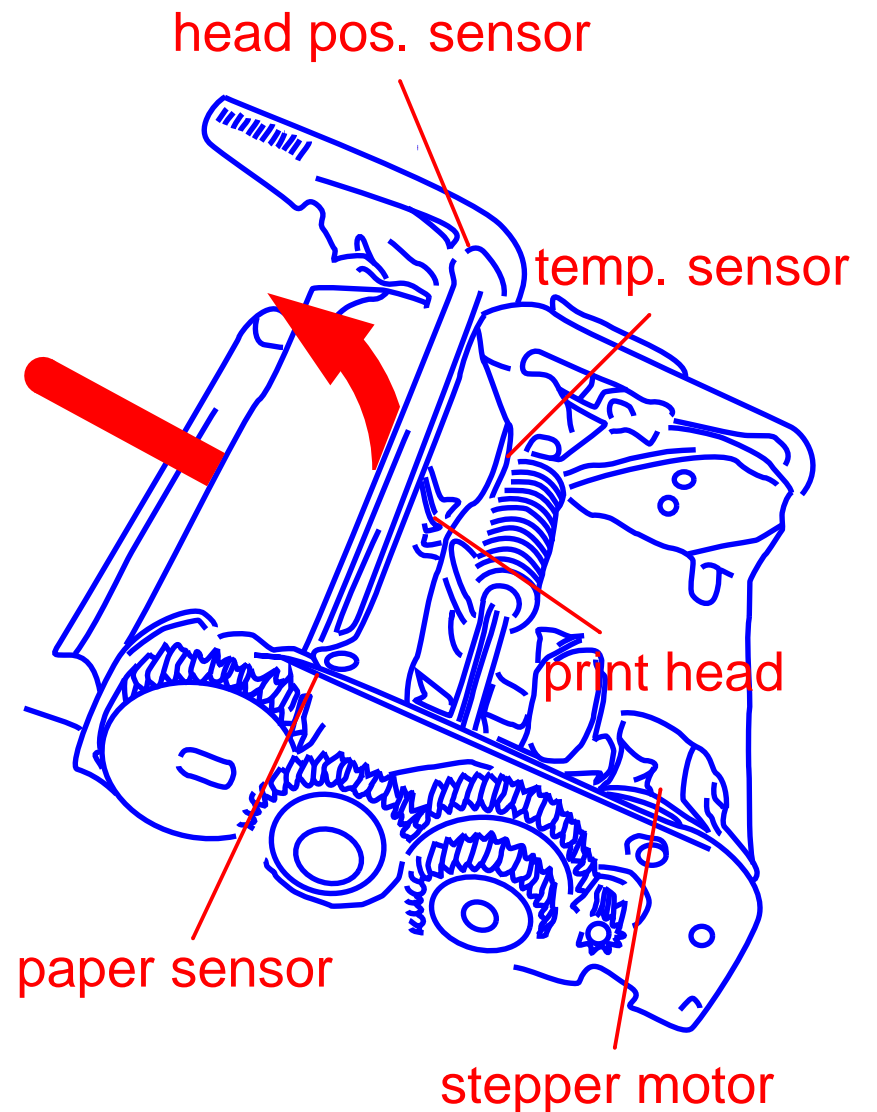
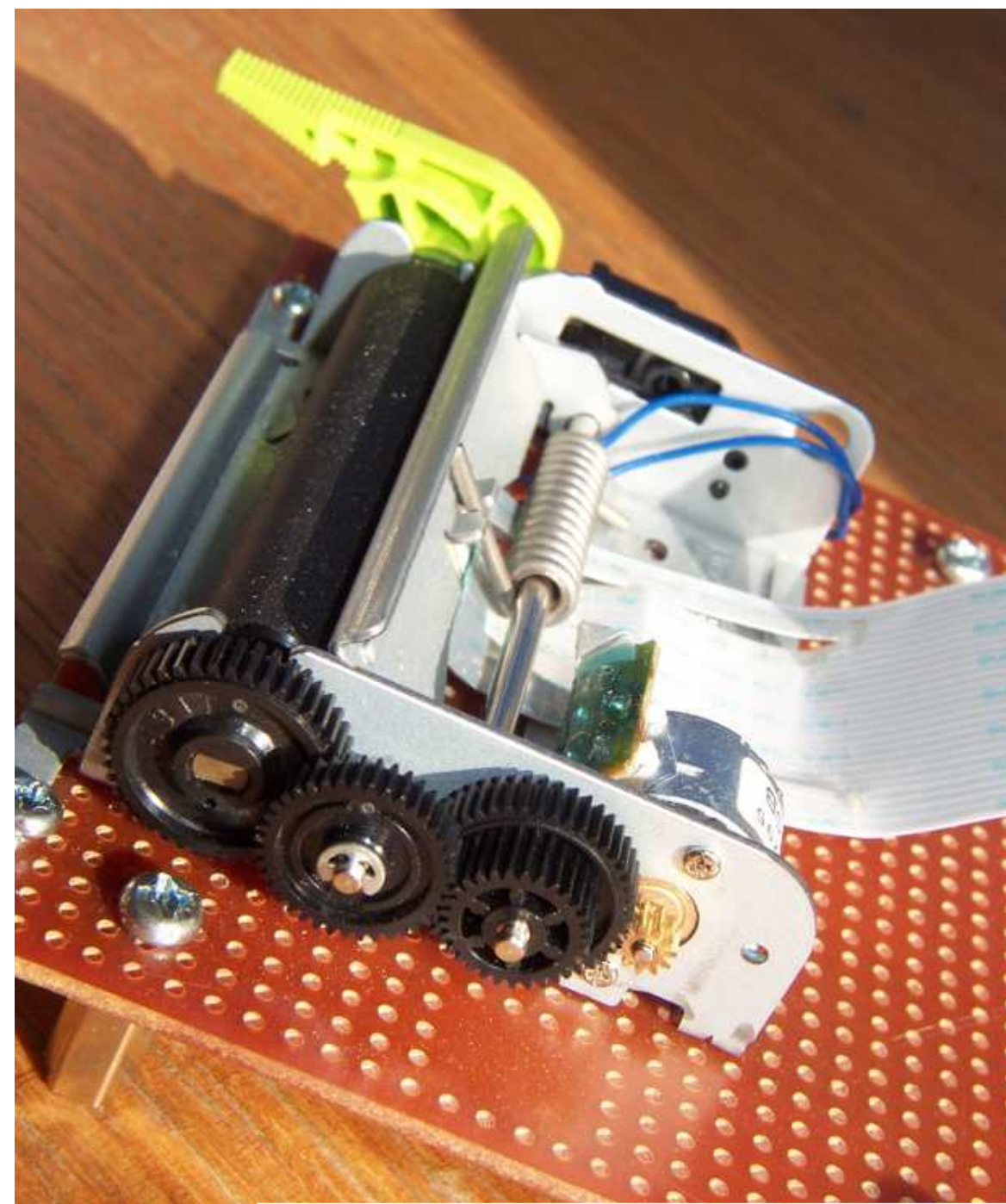
School of Computer Science and Engineering  
University of New South Wales, Sydney  
and NICTA

## Arcot Sowmya

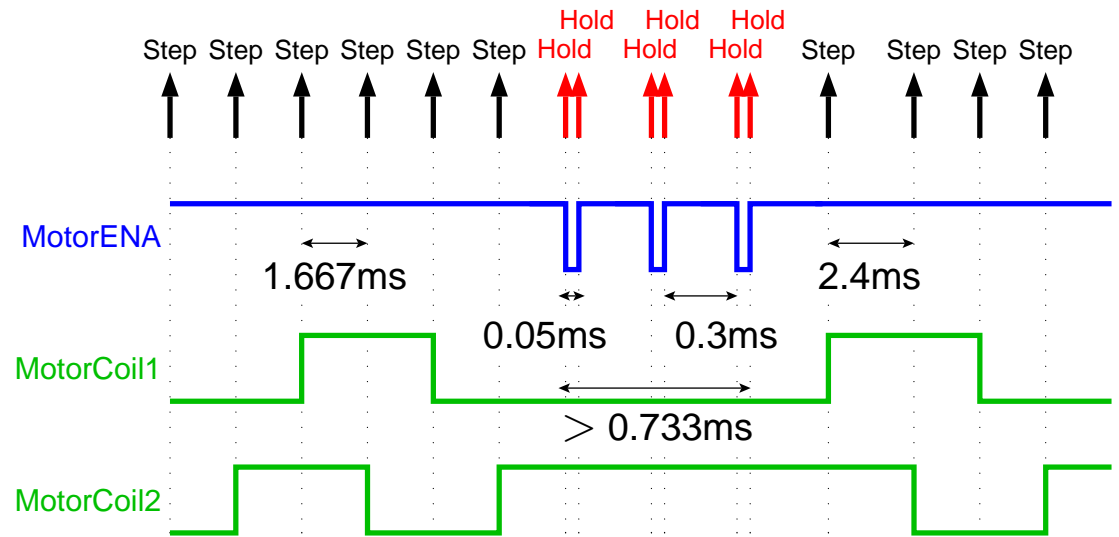
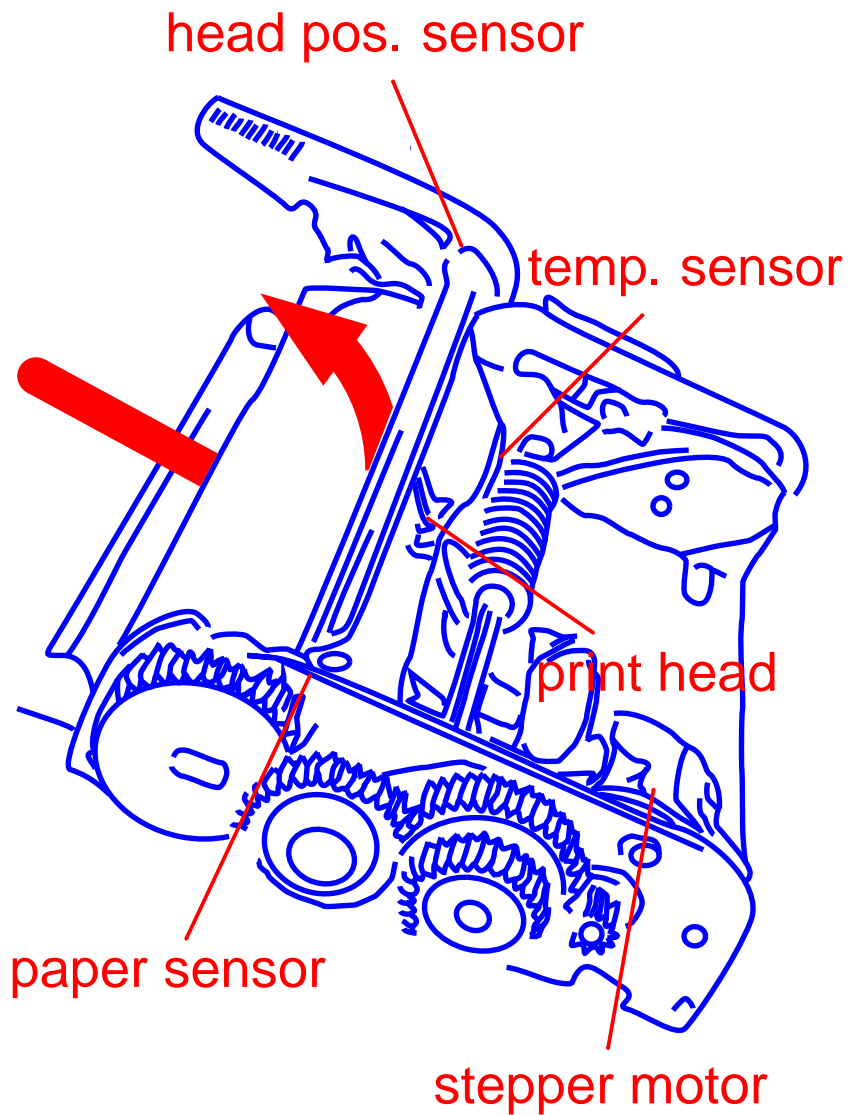
School of Computer Science and Engineering

University of New South Wales, Sydney





**Mix of reactive behaviour and physical time**



- Control requires complex sequential logic
- Intertwining of discrete events and timing
- Move motor by energising coils
- Sometimes the motor must be held steady
- Current is stuttered to reduce heat
- After stuttering the step period may change

```

module PrintSteps:
input Hold; output Step, MotorENA : boolean;

signal LongStep in
loop
  emit Step;
  present LongStep
  then % delay 2.4ms
  else % delay 1.660ms
  end present;

  present Hold then
    trap Stalling in
      loop
        emit MotorENA(false);
        % delay 0.05ms
        present Hold else exit Stalling end;
        emit MotorENA(true);
        % delay 0.3ms
        present Hold else exit Stalling end
      end loop
    ||
    % delay 0.733ms
    sustain LongStep
  end trap
end present
end loop
end signal

```

```

module STEPPER:
input Step;
output
  MotorCoil1 := true : boolean;
  MotorCoil2 := true : boolean;

loop
  await Step;
  emit MotorCoil1(false);

  await Step;
  emit MotorCoil2(false);

  await Step;
  emit MotorCoil1(true);

  await Step;
  emit MotorCoil2(true)
end loop

end module

```

## How should the delays be expressed?

- Pause statements?
- Timing inputs? (multiform time)
- External timers? ([CDO96, JPO95, MS92, BCG<sup>+</sup>97])
- External intervals? (CRP, [BRS93])
- Non-deterministic pause? (TAXYS, [BCP<sup>+</sup>01, STY03])
- Quantitative watchdogs? (Argos, [JMO93])

```
input Hold; output Step, MotorENA : boolean;
```

```
signal LongStep in
```

```
loop
```

```
  emit Step;
```

```
  present LongStep
```

```
  then await 48 tick;           % 2.4ms
```

```
  else await 32 tick;          % 1.660ms
```

```
  end present;
```

```
present Hold then
```

```
  trap Stalling in
```

```
    loop
```

```
      emit MotorENA(false);
```

```
      pause;                       % 0.05ms
```

```
      present Hold else exit Stalling end;
```

```
      emit MotorENA(true);
```

```
      await 6 tick;                 % 0.3ms
```

```
      present Hold else exit Stalling end
```

```
    end loop
```

```
  ||
```

```
    await 15 tick;                  % 0.733ms
```

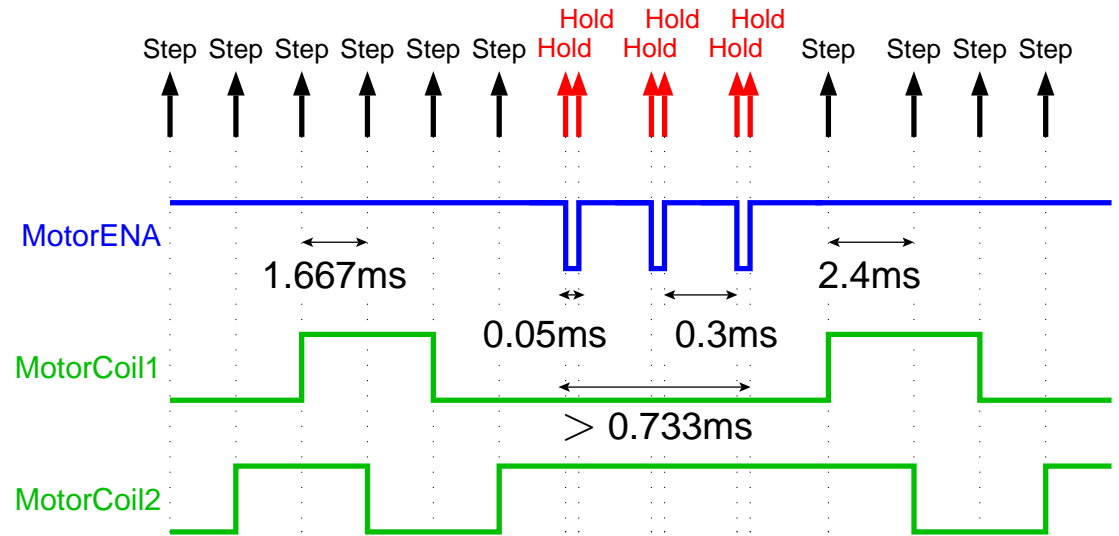
```
    sustain LongStep
```

```
  end trap
```

```
end present
```

```
end loop
```

```
end signal
```



## Attempt 1: fix period at 0.05ms

- **implies** sample-driven **implementation**
- forced to resolve spec. and impl.
- some delays are now approximate
- *not* an ideal specification; *not* portable
- meaning in terms of physical time?

```
input Hold; output Step, MotorENA : boolean;
```

```
signal LongStep in
```

```
loop
```

```
  emit Step;
```

```
  present LongStep
```

```
  then await 240 TNS; % 2.4ms
```

```
  else await 166 TNS; % 1.660ms
```

```
  end present;
```

```
present Hold then
```

```
  trap Stalling in
```

```
    loop
```

```
      emit MotorENA(false);
```

```
      await 5 TNS; % 0.05ms
```

```
      present Hold else exit Stalling end;
```

```
      emit MotorENA(true);
```

```
      await 30 TNS; % 0.3ms
```

```
      present Hold else exit Stalling end
```

```
    end loop
```

```
  ||
```

```
    await 73 TNS; % 0.730ms
```

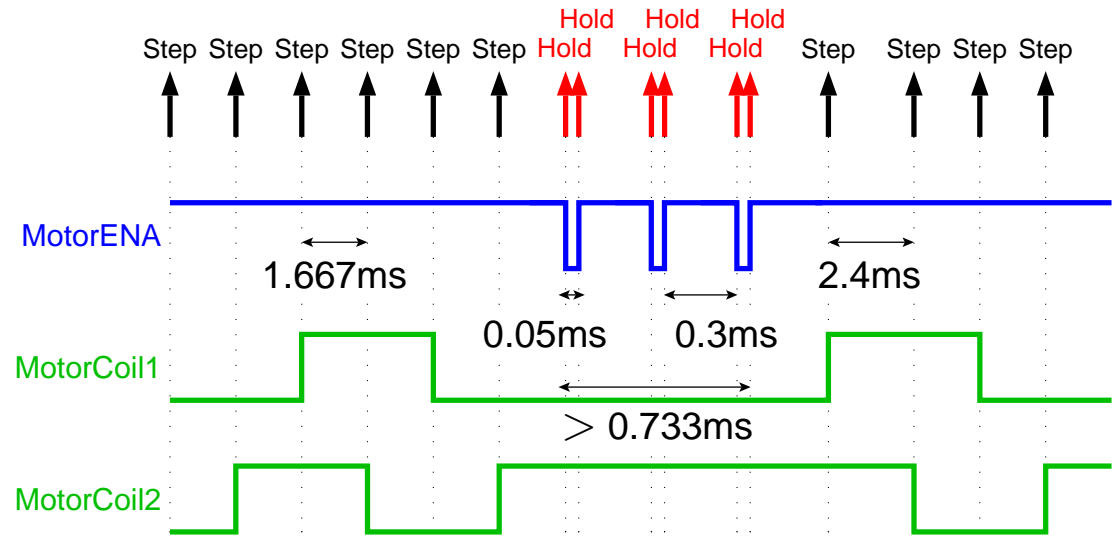
```
    sustain LongStep
```

```
  end trap
```

```
end present
```

```
end loop
```

```
end signal
```



## Attempt 2: TNS signal every 10ns

- Introduce suggestive signal names (adhoc)
- *Better* but not ideal
- Mandates a minimum execution rate
- Approximates delays (cannot delay trade)
- Granularity effects...

```

loop
  emit MotorENA(false);
  await 5 HMS;
  emit MotorENA(true);
  await 30 HMS
end loop

```

*or*

```

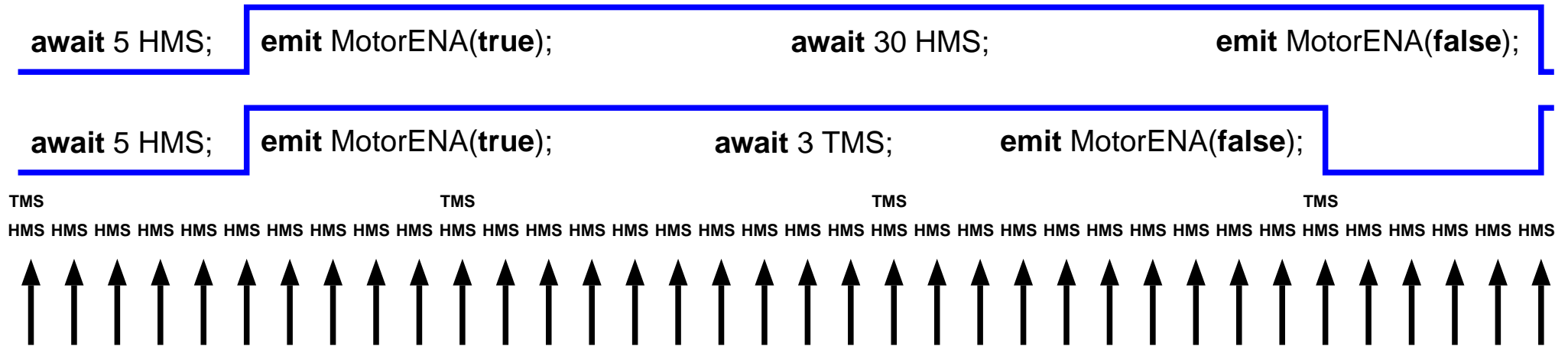
loop
  emit MotorENA(false);
  await 5 HMS;
  emit MotorENA(true);
  await 3 TMS
end loop

```

Assume HMS every 0.01ms, TMS every 0.10ms.

Signal counts are relative to system startup  
*not* statement initialization.

Even given **relation**  $TMS \Rightarrow HMS$ ,  
exchanging 30 HMS for 3 TMS changes the delay.



loop

```
emit MotorENA(false);
await 5 HMS;
emit MotorENA(true);
await 30 HMS
```

end loop

or

loop

```
emit MotorENA(false);
await 5 HMS;
emit MotorENA(true);
await 3 TMS
```

end loop

Assume HMS every 0.01ms, TMS every 0.10ms.

Signal counts are relative to system startup  
*not* statement initialization.

Even given **relation**  $TMS \Rightarrow HMS$ ,  
exchanging 30 HMS for 3 TMS changes the delay.

```
input Hold; output Step, MotorENA : boolean;
```

```
signal LongStep in
```

```
loop
```

```
  emit Step;
```

```
  present LongStep
```

```
  then emit START_T1(2400); await T1 % 2.4ms
```

```
  else emit START_T1(1667); await T1 % 1.667ms
```

```
  end present;
```

```
present Hold then
```

```
  trap Stalling in
```

```
    loop
```

```
      emit MotorENA(false);
```

```
      emit START_T1(50); await T1; % 0.05ms
```

```
      present Hold else exit Stalling end;
```

```
      emit MotorENA(true);
```

```
      emit START_T1(300); await T1; % 0.3ms
```

```
      present Hold else exit Stalling end
```

```
    end loop
```

```
  ||
```

```
    emit START_T1(733); await T1; % 0.733ms
```

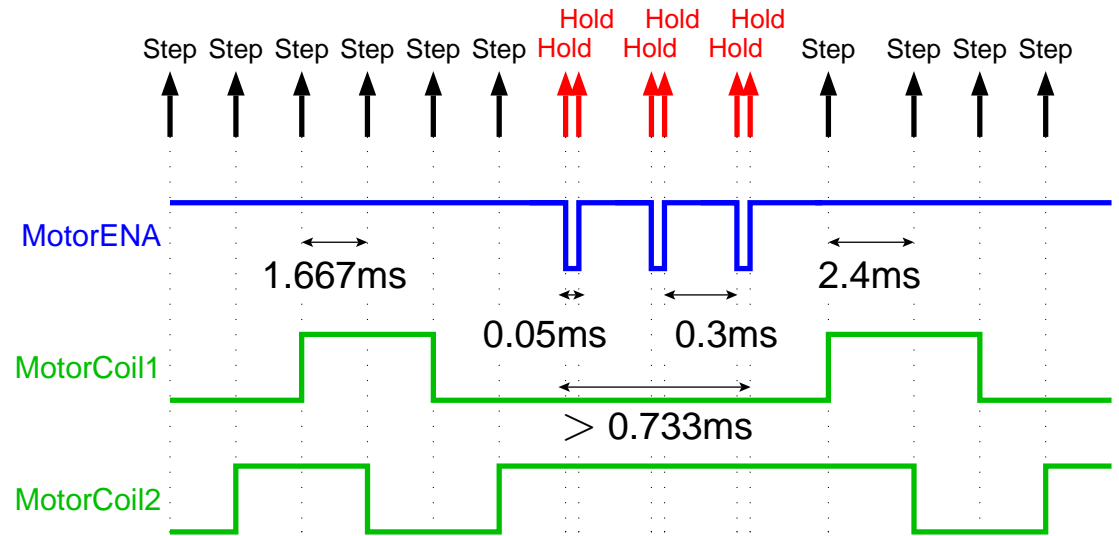
```
    sustain LongStep
```

```
  end trap
```

```
end present
```

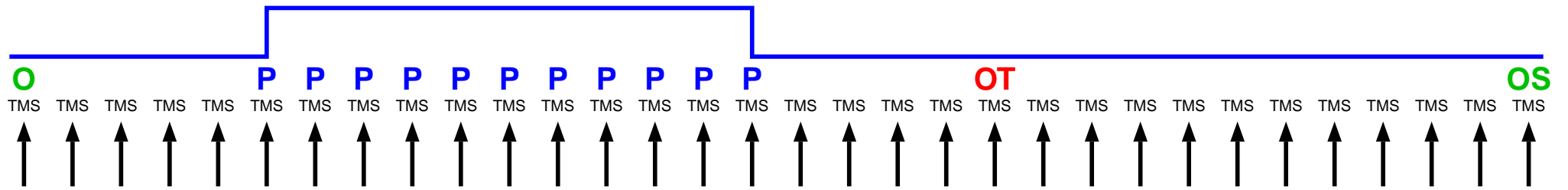
```
end loop
```

```
end signal
```



### Attempt 3: One-shot timers with ns resolution

- Natural for embedded engineers
- Finer granularities possible
- Shift from meaning to mechanism
- Harder to analyze and manipulate
- Does not interact well with suspension...

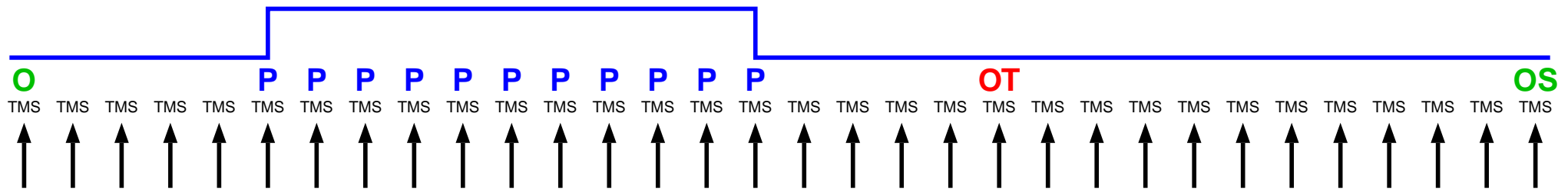


```

emit O;
suspend
  await 20 TMS;
  emit OS
||
  emit START_TMSTIMER01(20);
  await TMSTIMER01;
  emit OT
when P

```

**Timers are incompatible with suspension**



```

emit O;
suspend
  await 20 TMS;
  emit OS
||
emit START_TMSTIMER01(20);
await TMSTIMER01;
emit OT
when P

```

```

emit O;
suspend
  await 20 TMS;
  emit OS
||
exec HMSTIMER01(20);
emit OT
when P

```

[BRS93, Ber93]

**Timers are incompatible with suspension**

```
input Hold; output Step, MotorENA : boolean;
```

```
signal LongStep in
```

```
loop
```

```
  emit Step;
```

```
  present LongStep
```

```
  then delay 2.4ms % 2.4ms
```

```
  else delay 1.667ms % 1.667ms
```

```
  end present;
```

```
present Hold then
```

```
  trap Stalling in
```

```
    loop
```

```
      emit MotorENA(false);
```

```
      delay 0.05ms % 0.05ms
```

```
      present Hold else exit Stalling end;
```

```
      emit MotorENA(true);
```

```
      delay 0.3ms; % 0.3ms
```

```
      present Hold else exit Stalling end
```

```
    end loop
```

```
  ||
```

```
    delay 0.733ms; % 0.733ms
```

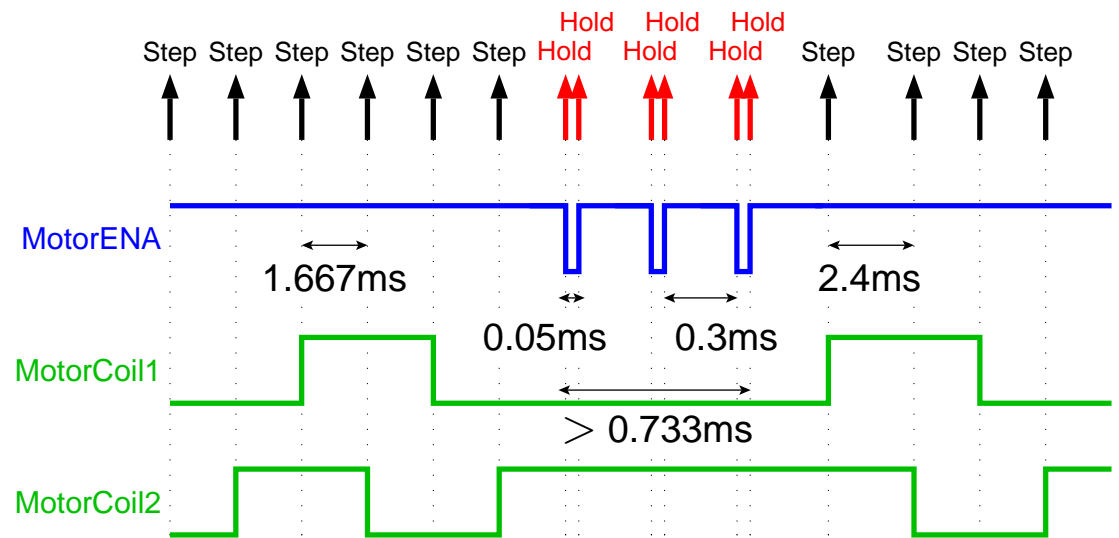
```
    sustain LongStep
```

```
  end trap
```

```
end present
```

```
end loop
```

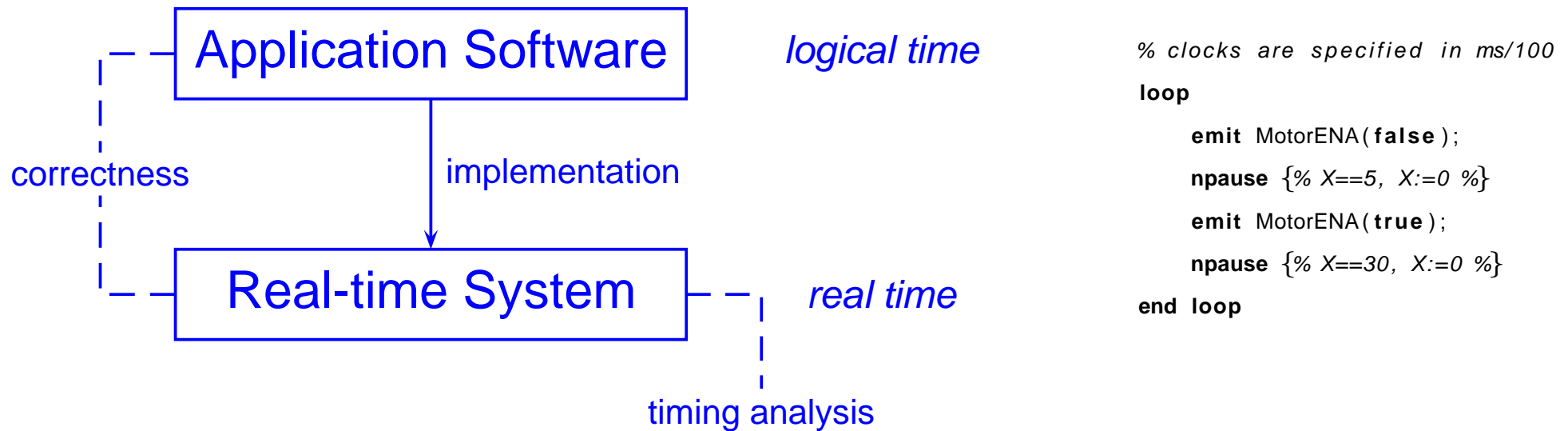
```
end signal
```



## The Solution: real-time delay statement

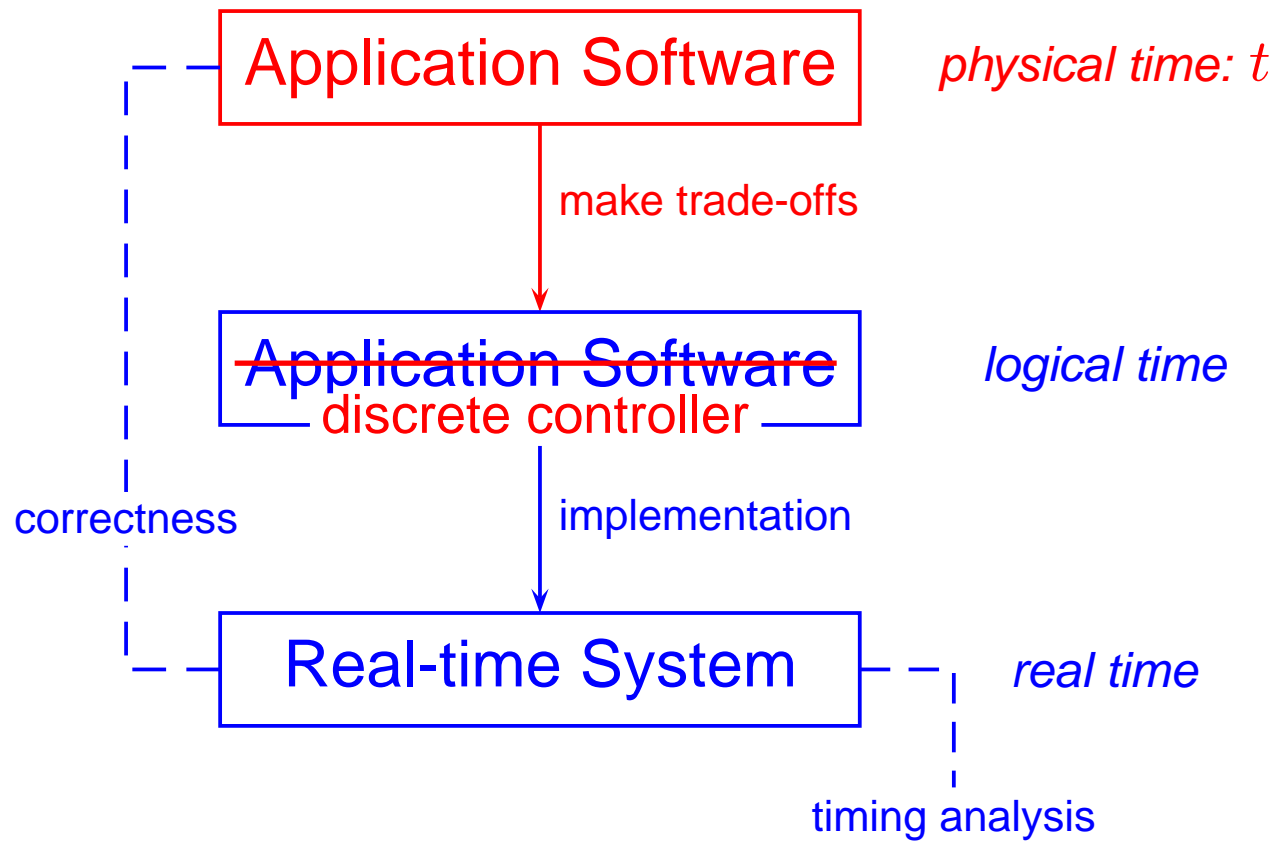
- Simple idea: Why not state delays directly in real-time?
- Verify & simulate in terms of model/spec. (not lots of clicking, more like in Uppaal)
- Postpone implementation choices...

# TAXYS [STY03]



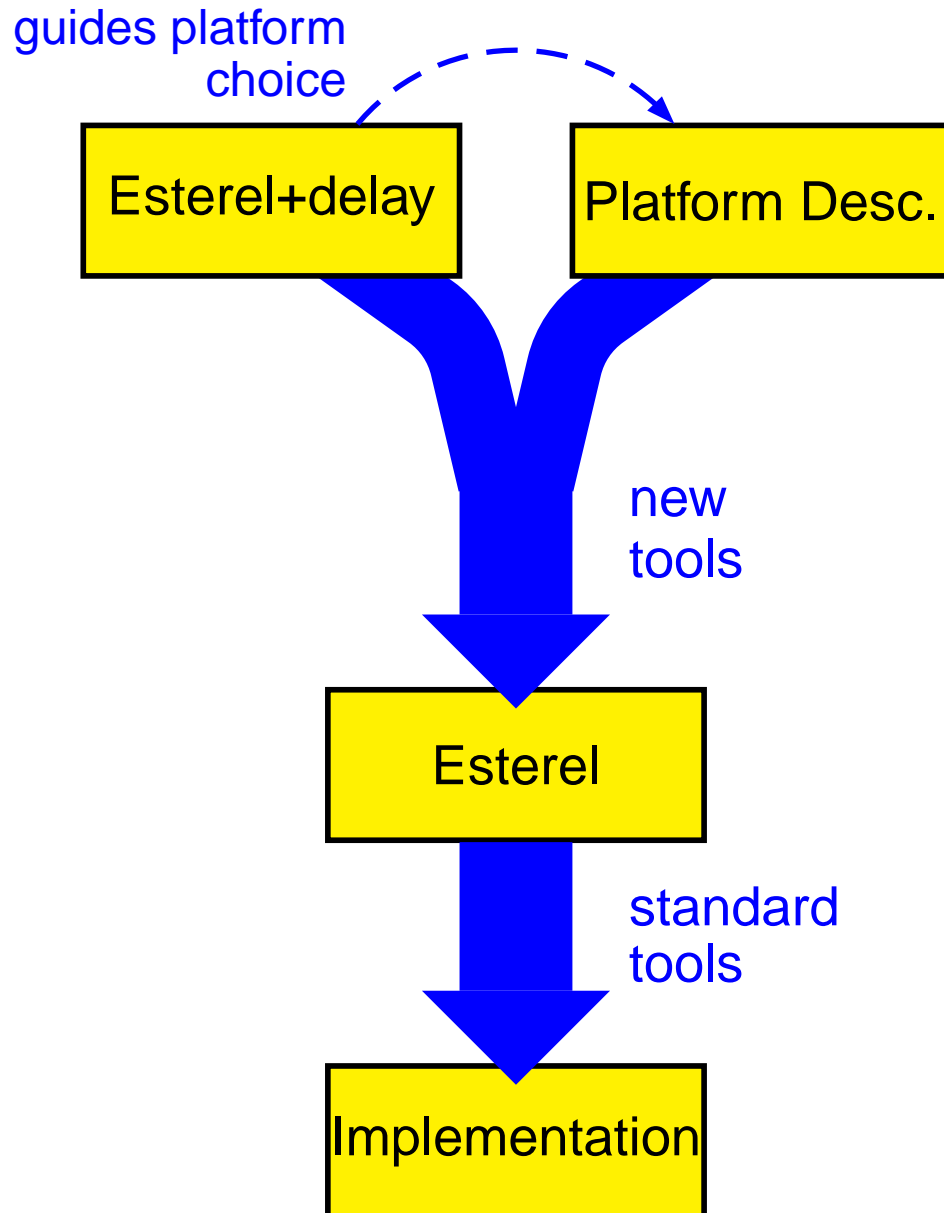
“Whenever event  $a$  occurs, event  $b$  will follow at most after  $x$  time units.” This is a *nonfunctional* property, because the time units are measured in *real-time*.

**Time is both a resource and a behavioural dimension.**



**Multiform time is not always appropriate**

# Esterel+delay



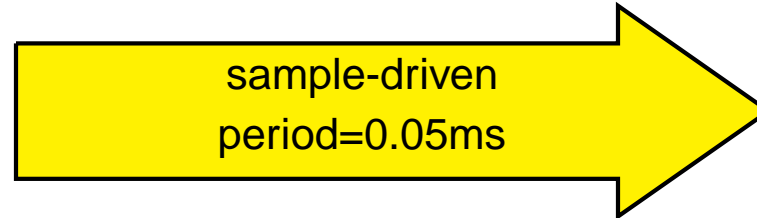
- Esterel + **delay** statement
- Program guides choice of platform
- Describe platform abstractly
- Convert Esterel+delay program and platform description into an Esterel program...
- ... and compile with standard tools

# Esterel+delay: examples

```

loop
  emit motorENA(false);
  delay 0.05ms;
  emit motorENA(true);
  delay 0.3ms;
end loop

```



```

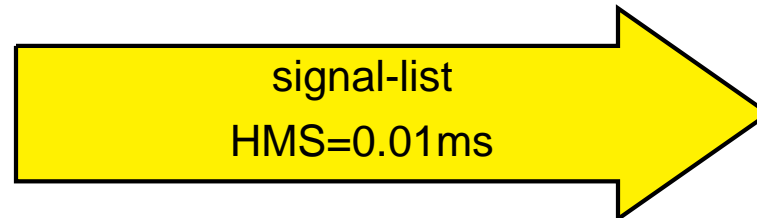
loop
  emit motorENA(false);
  pause;
  emit motorENA(true);
  await 6 tick;
end loop

```

```

loop
  emit motorENA(false);
  delay 0.05ms;
  emit motorENA(true);
  delay 0.3ms;
end loop

```



```

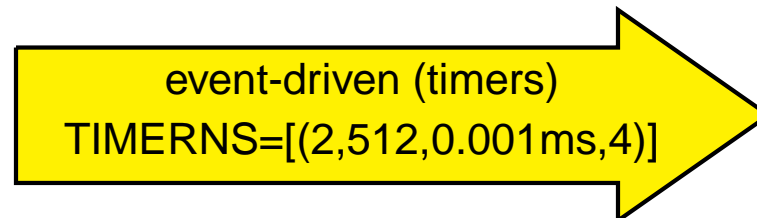
loop
  emit motorENA(false);
  await 5 HMS;
  emit motorENA(true);
  await 30 HMS;
end loop

```

```

loop
  emit motorENA(false);
  delay 0.05ms;
  emit motorENA(true);
  delay 0.3ms;
end loop

```



```

loop
  emit motorENA(false);
  emit TIMERNS1(50);
  await TIMERNS1;
  emit motorENA(true);
  emit TIMERNS1(300);
  await TIMERNS1;
end loop

```

# Implementation: sample-driven

$$\tau \in \mathbb{Q}^{>0}$$

if  $d = n \cdot \tau$ ,

$\mathcal{T}_\tau(\mathbf{delay}_{approx} d) = \mathbf{await} n \text{ tick}$ ,

otherwise,

$\mathcal{T}_\tau(\mathbf{delay}_{under} d) = \mathbf{await} l \text{ tick}$ , and

$\mathcal{T}_\tau(\mathbf{delay}_{over} d) = \mathbf{await} u \text{ tick}$ ,

and, when  $d \cdot \tau \geq 1$ ,

$\mathcal{T}_\tau(\mathbf{delay}_{avg} d) = \mathbf{if} \text{ abs}(\text{diff} + d_l) \leq \text{abs}(\text{diff} - d_u)$   
 $\quad \mathbf{then} \text{ diff} = \text{diff} + d_l$ ;  
 $\quad \mathbf{await} l \text{ tick}$   
 $\quad \mathbf{else} \text{ diff} = \text{diff} - d_u$ ;  
 $\quad \mathbf{await} u \text{ tick}$   
 $\mathbf{end if}$ ,

$$l = \max \left( \left\lfloor \frac{d}{\tau} \right\rfloor, 1 \right) \quad (1)$$

$$u = \left\lceil \frac{d}{\tau} \right\rceil \quad (2)$$

# Implementation: counting events

 $(s, \tau_s)$ 

- Just choose  $s$  with smallest  $\tau_s$
- Or, try to account for discrepancy between relative and absolute delays...

```

delay 3;          +0          [...]
emit O1;         +3
loop
  emit O2;       +3  +10  +17  ...
  delay 2;       +3  +10  +17  ... [7, 3]
  emit O3;       +5  +12  +19  ...
  delay 5        +5  +12  +19  ... [7, 5]
end loop         +10  +17  +24  ...

```

Figure 6.8: Phase relationships in an Esterel+delay program

- ...and then, for each delay, choose the best  $s$
- Not really sure:
  - how to do this in general
  - whether it's useful or not

# Implementation: timers

 $(\tau_t, l, u, n)$ 

## Definition 6.4.6

A *delay term* is formed from constants in  $\mathbb{Q}^{\geq 0}$ , and the two binary operators ; and ||. ■

## Definition 6.4.7

The *delay abstraction* function  $D$  maps an Esterel+delay program, where delay expressions have been evaluated, to a delay term:

$$D(\mathbf{nothing}) = 0$$

$$D(\mathbf{emit } s) = 0$$

$$D(\mathbf{pause}) = 0$$

$$D(\mathbf{delay } d) = d$$

$$D(\mathbf{present } s \mathbf{ then } p \mathbf{ else } q \mathbf{ end}) = D'(p, q, ;)$$

$$D(\mathbf{suspend } p \mathbf{ when } s \mathbf{ end}) = D(p)$$

$$D(p ; q) = D'(p, q, ;)$$

$$D(\mathbf{loop } p \mathbf{ end}) = D(p)$$

$$D(p || q) = D'(p, q, ||)$$

$$D(\mathbf{trap } t \mathbf{ in } p \mathbf{ end}) = D(p)$$

$$D(\mathbf{exit } t) = 0$$

$$D(\mathbf{signal } s \mathbf{ in } p \mathbf{ end}) = D(p)$$

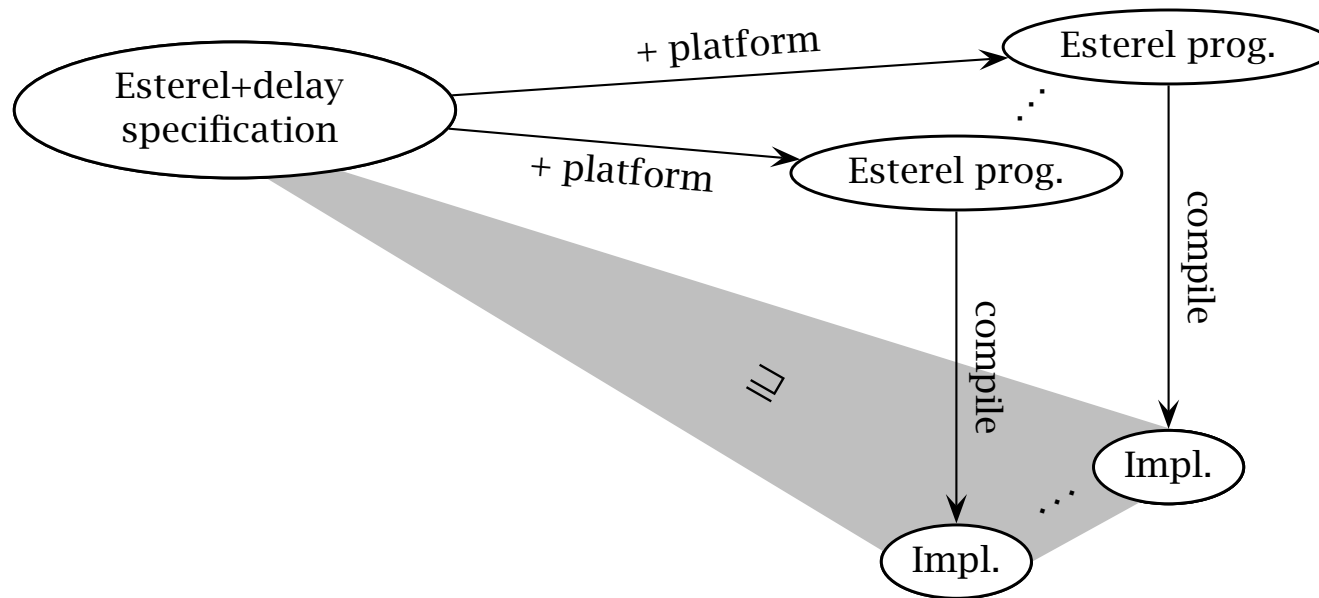
- Form abstraction from program:
  - simultaneous delays (||)
  - sequential delays (;)
- Take set of timers  $(\tau_t, l, u, n)$
- problem: allocate timers to delays
- Manual or (semi-)automatic?

where:

$$D'(p, q, \otimes) = \begin{cases} D(q) & \text{if } D(p) = 0 \\ D(p) & \text{if } D(q) = 0 \\ D(p) \otimes D(q) & \text{otherwise.} \end{cases}$$

■

# Semantics, fuzziness, ... deadends?



- Need some notion of implementation with ‘**fuzziness**’ [BS08]
- Not just a transformation to timed automata for the sake of it...
- ... but something that improves the engineering aspects
- Not clear how to proceed beyond shallow syntactic transformations
- Need to mix **instantaneousness** with **duration**
- **Can it be done without losing the essence of Esterel?**

# References

- [BCG<sup>+</sup>97] Felice Balarin, Massimiliano Chiodo, Paolo Giusto, Harry Hsieh, Attila Jurecska, Luciano Lavagno, Claudio Passerone, Alberto Sangiovanni-Vincentelli, Ellen Sentovich, Kei Suzuki, and Bassam Tabbara. *Hardware-Software Co-design of Embedded Systems: The POLIS Approach*. The Kluwer International Series in Engineering and Computer Science. Kluwer Academic Publishers, 1997.
- [BCP<sup>+</sup>01] V. Bertin, E. Closse, M. Poize, J. Pulou, J. Sifakis, P. Venier, D. Weil, and S. Yovine. Taxys = Esterel + Kronos: A tool for verifying real-time properties of embedded systems. In *Proceedings of 40th IEEE Conference on Decision and Control*, pages 2875–2880, Orlando, Florida, USA, December 2001. IEEE.
- [Ber93] Gérard Berry. Preemption in concurrent systems. In R. K. Shyamasundar, editor, *Foundations of Software Technology and Theoretical Computer Science*, volume 761 of *Lecture Notes in Computer Science*, Bombay, India, December 1993. Springer-Verlag.
- [BRS93] G. Berry, S. Ramesh, and R. K. Shyamasundar. Communicating reactive processes. In *Proceedings of 20th ACM SIGPLAN-SIGACT Symposium on Principles Of Programming Languages (POPL 1993)*, pages 85–98. ACM Press, 1993.
- [BS08] Henrik Bohnenkamp and Mariëlle Stoelinga. Quantitative testing. In *Proceedings of 8th ACM*

*International Conference on Embedded Software (EMSOFT'08)*, pages 227–236, Atlanta, Georgia USA, October 2008. ACM, ACM Press.

- [CDO96] Claude Castelluccia, Walid Dabbous, and Sean O'Malley. Generating efficient protocol code from an abstract specification. *ACM SIGCOMM Computer Communication Review*, 26(4):60–72, October 1996.
- [JMO93] M. Jourdan, F. Maraninchi, and A. Olivero. Verifying quantitative real-time properties of synchronous programs. In Costas Courcoubetis, editor, *5th International Conference on Computer Aided Verification*, volume 697 of *Lecture Notes in Computer Science*, pages 347–358, Elounda, Greece, June/July 1993. Springer-Verlag.
- [JPO95] Lalita Jategaonkar Jagadeesan, Carlos Puchol, and James E. Von Olnhausen. A formal approach to reactive systems software: A telecommunications application in Esterel. In *Proceedings of Workshop on Industrial-Strength Formal Specification Techniques*, pages 132–145, Florida, USA, April 1995. IEEE.
- [MS92] Gary J. Murakami and Ravi Sethi. Parallelism as a structuring technique: Call processing using the Esterel language. In J. van Leeuwen, editor, *Proceedings of 12th International Federation for Information Processing (IFIP) World Computer Congress*, number 92 in Information Processing, pages 10–16, Madrid, Spain, 1992.
- [STY03] Joseph Sifakis, Stavros Tripakis, and Sergio Yovine. Building models of real-time systems from

application software. *Proceedings of IEEE*, 91(1):100–111, January 2003.