Australian National University

Well, there's a dull ache, certainly.

And layered on top of that is a club sandwich of pain. Only instead of bacon, there's

Real Time Systems

ENGN8537

Embedded Systems

Outline

- What is time?
- What is real time (really)?
- Interfacing with time
- Safety and Liveness

References:
Burns and Wellings **Real-Time Systems and Programming Languages (Third Edition)**, Addison Wesley Longmain 2001

Thanks to Dr Uwe Zimmer for inspiration and material from COMP4330

What is Time?

Do we exist in time, or is it part of our existence?

Is time an intrinsic property of nature? (Platonism)
Time is an underlying external phenomenon. Thus simultaneous events happen at the same exact time. There is an underlying assumption that time is progressing, even if no changes can be observed.

Is time a construct which is based on external events? (Reductionism)
Time is an observation of distinguishable events. If all possible observers detect one event before another, they are said to be in sequence. If this cannot be assumed for all observers, they are said to be simultaneous. Therefore the notion of time is actually one of causality.

.. but what happens to time between events?

What is Time?

Is time an intrinsic property of nature?
Quantum Physics. Time is a parameter of the quantum wavefunction from which all state is derived.

Is time a construct which is based on external events?
General Relativity. Time and Space are the same, distinguished from each other by the sign of the diagonal entries of the Metric Tensor. If two events can be connected by a "straight trajectory" in time and space where the velocity is less than light, the events are "time-like", and are separated in time, but not necessarily in space (for all observers). If the velocity of the path between events is greater than the speed of light, the events are "space-like" in that they're definitely separated in space, but might happen at the same time for some observer. This implies that time(space) and events in time(space) are not independent.

What is Time?

What is time Mathematically?

1. Transitivity: $x < y \land y < z \Rightarrow x < z$
2. Linearity: $x < y \lor x > y \Rightarrow x \neq y$
3. Irreflexivity: $\text{not}(x < x)$
4. Density: $x < y \Rightarrow \exists z \mid x < z \land z < y$

The density condition states that time is continuous. Any *measurement* of time however is necessarily discrete.

What is Time?

UT0	1884: Mean solar time at Greenwich meridian
UT1	Corrected UT0, ☞ polar motion
UT2	Corrected UT1, ☞ variations in the speed of rotation of the earth
IAT (International Atomic Time)	a caesium 133 atomic clock (current accuracies: one miss in 10^{13} ticks, e.g. approximately once every 300000 years)
UTC (Universal Time Coordinated)	a IAT clock, which is synchronized to UT2 (by introducing occasional leap ticks) – difference between UTC and IAT is < 0.5 s
‘1 sec.’	1/86400 of a mean solar day … or … 1/31566925.9747 of the tropical year for 1900 (Ephemeris Time defined 1955) … or … 9192631770 periods of the radiation corresponding to the transition between two hyperfine levels of the ground state of the caesium 133 atom

What is Time?

Fundamentally, time is tracked in an Embedded System by a series of logic level transitions with well defined periods (Clocks), typically derived from the vibration of precisely cut pieces of crystal (Crystal Oscillators, XTALs) or the repeated buildup and release of charge from a capacitor (RC Oscillator).

There may be a hierarchy of such sources, each one corrected by another. For example, a high frequency crystal phase- or frequency-locked to a high precision timing device.

You'll often come across the concept of "Clock Skew" in FPGAs. Clock Skew errors come about because the FPGA's concept of "Now" is linked to a clock edge, but the finite propagation speed of that edge means not every part of the chip has the same idea of "Now"!

Input clocks are the only fundamental concept of time that a processor can understand, but is it Real?

What is Real Time?

In practical Engineering, the precise definition and understanding of time isn't often required.

With so many different definitions of time though, which one is Real?

Time in an Embedded System is given with reference to an external source. It doesn't necessarily matter which source, so long as it's consistent though the design.

The most practical time source is selected for the task and is used as the Real Time.

What is Embodiment?

Hypothesis

Embodied Phenomena are those that by their very nature, occur in real time and real space.

Refinement

Embodiment is the property of any engagement with the real world that (may) may that engagement meaningful. (Paul Dourish)

Claim

Embodied Phenomena are the essence of meaningful interaction.
Embedded Systems are Technical Instantiations of Embodied Phenomena.

What is Embodiment?

Meaningful Embedded Systems are part of an ‘ecological niche’ (Rolf Pfeifer)

- The Operational Environment is supportive of, and supported by the system
- The Embedded System is constructed as part of the Operational Environment according to the task
- The task is meaningful considering the morphology and cognitive ability of the system as well as the response of the Environment.

Situated, Embodied and Self-Sufficient.

What is Embodiment?

Embodied skills depend on a tight coupling between perception and action, up to the level where the distinction between each can be difficult.

Tight coupling between perception and action means to operate under real time constraints (and to construct meaningful morphologies).

An Embedded System operates under constraints on real time. To make its interaction with the world meaningful, such a system must depend not only on the logical correctness of a result, but the real time at which that result was delivered.

Real Time Systems

A Real Time system depends not only on the logical correctness of a result, but the time at which that result was delivered.

Not too early, not too late. A ‘Goldilocks’ result.

Often defined in terms of a deterministic response to an external stimulus

- Car crash ➔ Airbag deployment: An airbag that deploys a second too late is worse than no airbag at all!

Real Time Systems

A Real Time system depends not only on the logical correctness of a result, but the time at which that result was delivered.

Note that we haven’t referred to the system speed. This is intentional, contrary to popular belief, real time systems are not necessarily fast, just predictable (predictably correct).

In general, a non-RT system cannot be made to obey a RT specification by making it faster.

Real Time

Motivation

- Failing to process inputs from a limit switch in a bounded time
  - your robot rips itself apart
- Failing to process sensor measurements in a bounded time
  - your jet engine blows a hole through the plane and its passengers
- Failing to process an iterative control loop with a constant period
  - your plant becomes unstable and dumps chemical sludge on an Auklet

Real Time Constraints

Classification

Hard	Single failure leads to severe malfunction
Firm	Results are meaningless after the deadline
	Only multiple or permanent failures threaten the whole system
Soft	Results may still be useful after the deadline

This is a useful classification, however the distinction is rarely this obvious in real systems. We will come to the specification of timing constraints a little later in the lecture.

Identifying Real Time Constraints

Case Study Airplane

1. Jet control systems
2. Airfoil control systems
3. Instrumentation
4. Wireless Communications
5. Lighting and comfort systems
6. Entertainment systems

Identifying Real Time Constraints

Case Study Airplane

1. Jet control systems Hard! The controller has a well defined time during which it must react to stimuli such as over-temperature conditions before the system will fail
2. Airfoil control systems
3. Instrumentation
4. Wireless Communications
5. Lighting and comfort systems
6. Entertainment systems

Identifying Real Time Constraints

Case Study Airplane

1. Jet control systems Hard
2. Airfoil control systems Hard! A delay changes the stability of a control system – a varying delay will change the tuning of that system
3. Instrumentation
4. Wireless Communications
5. Lighting and comfort systems
6. Entertainment systems

Identifying Real Time Constraints

Case Study Airplane

1. Jet control systems Hard
2. Airfoil control systems Hard
3. Instrumentation Hard/Firm. The instrumentation as connected to the autopilot systems will be hard for the same reasons as the airfoil controls. The display of the values to the pilot is probably not hard real time, so long as it doesn't exceed some (large-to-a-computer) value
4. Wireless Communications
5. Lighting and comfort systems
6. Entertainment systems

Identifying Real Time Constraints

Case Study Airplane

1. Jet control systems Hard
2. Airfoil control systems Hard
3. Instrumentation Hard/Firm
4. Wireless Communications Firm/Soft: The underlying protocol is generally hard RT as the timing of chips, frames etc. is part of the protocol specification. The data level cannot be deterministic, and therefore cannot be real time, in the general case!
5. Lighting and comfort systems
6. Entertainment systems

Identifying Real Time Constraints

Case Study Airplane

1. Jet control systems Hard
2. Airfoil control systems Hard
3. Instrumentation Hard/Firm
4. Wireless Communications Firm/Soft
5. **Lighting and comfort systems Firm/Soft:** The low level brightness control (PWM) is probably firm RT, pump control is firm RT, but the response time of light switch to action (for example) probably isn't
6. Entertainment systems

Identifying Real Time Constraints

Case Study Airplane

1. Jet control systems Hard
2. Airfoil control systems Hard
3. Instrumentation Hard/Firm
4. Wireless Communications Firm/Soft
5. Lighting and comfort systems Firm/Soft
6. Entertainment systems An embedded system in their own right! Probably not RT with respect to their integration in to the airplane system at least

A Real-Time Hierarchy

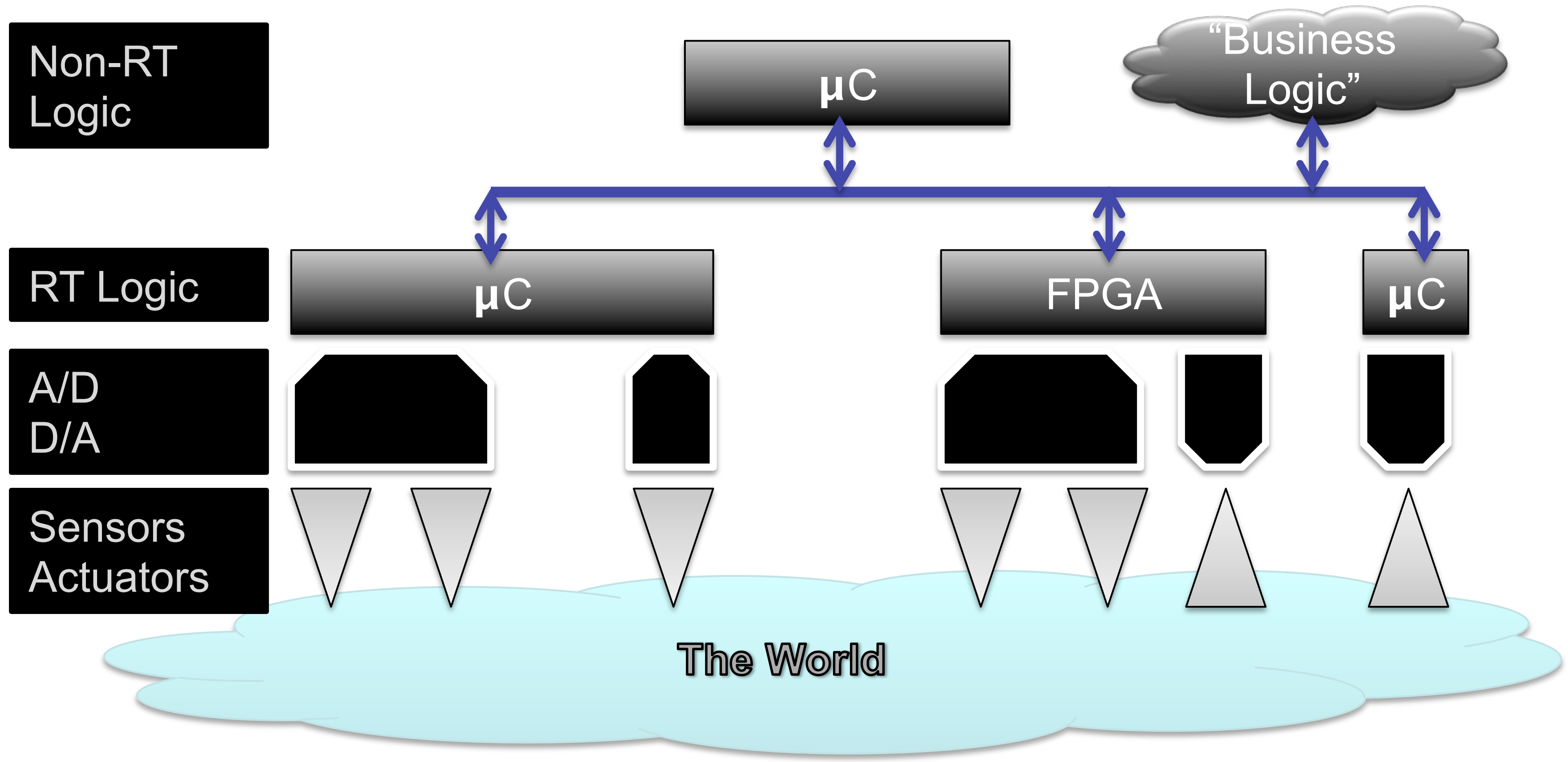
Many systems have a mix of components with and without RT constraints.

Often, the rigidity of the constraint is inversely proportional to the component complexity

- Sensor/Actuator driver: Hard RT Constraints, relatively simple
- Graphical User Interface: Few/No RT constraints, 10k – 100k lines of code

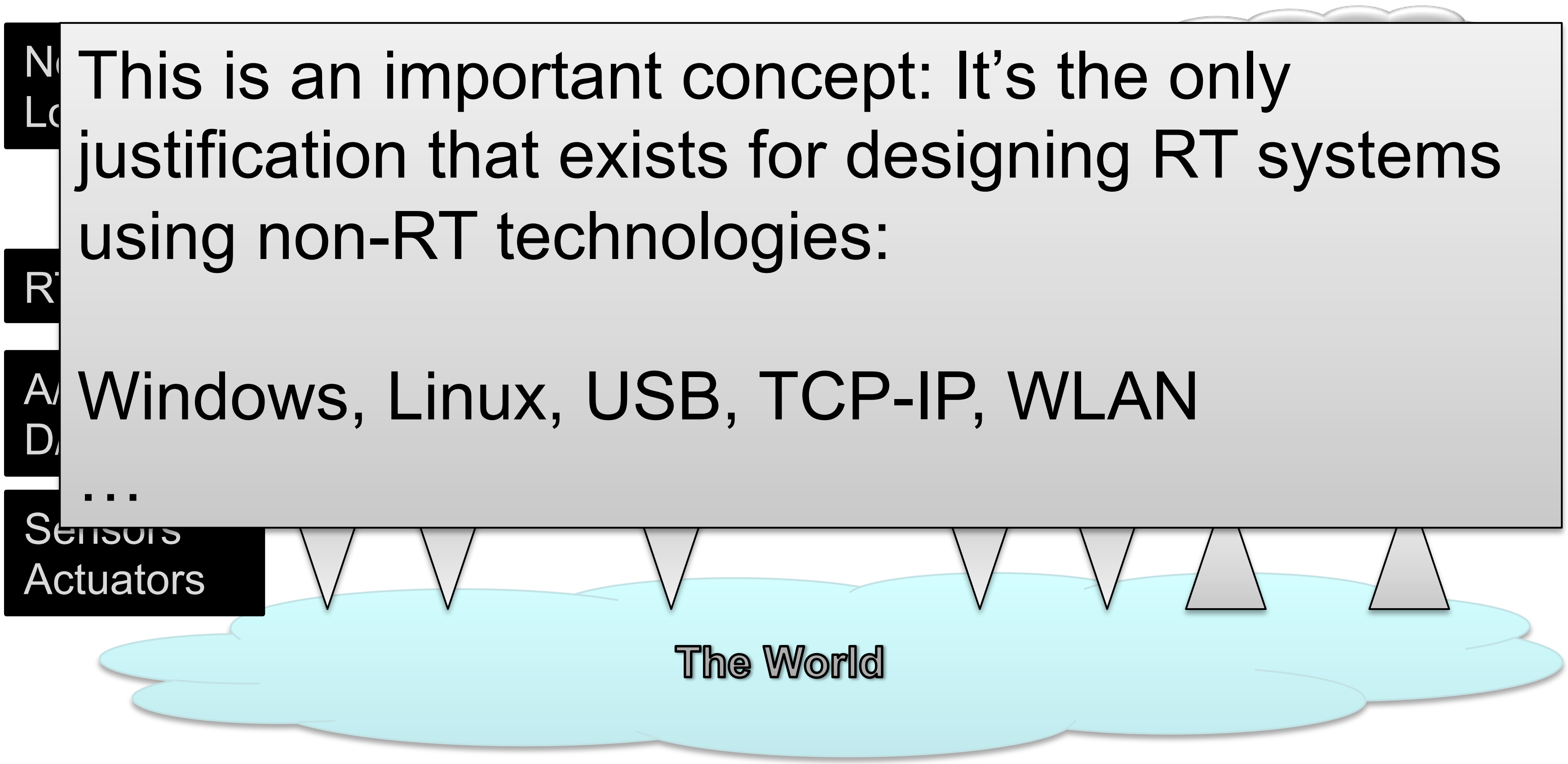
A Real-Time Hierarchy

This naturally forms a hierarchy



A Real-Time Hierarchy

This naturally forms a hierarchy



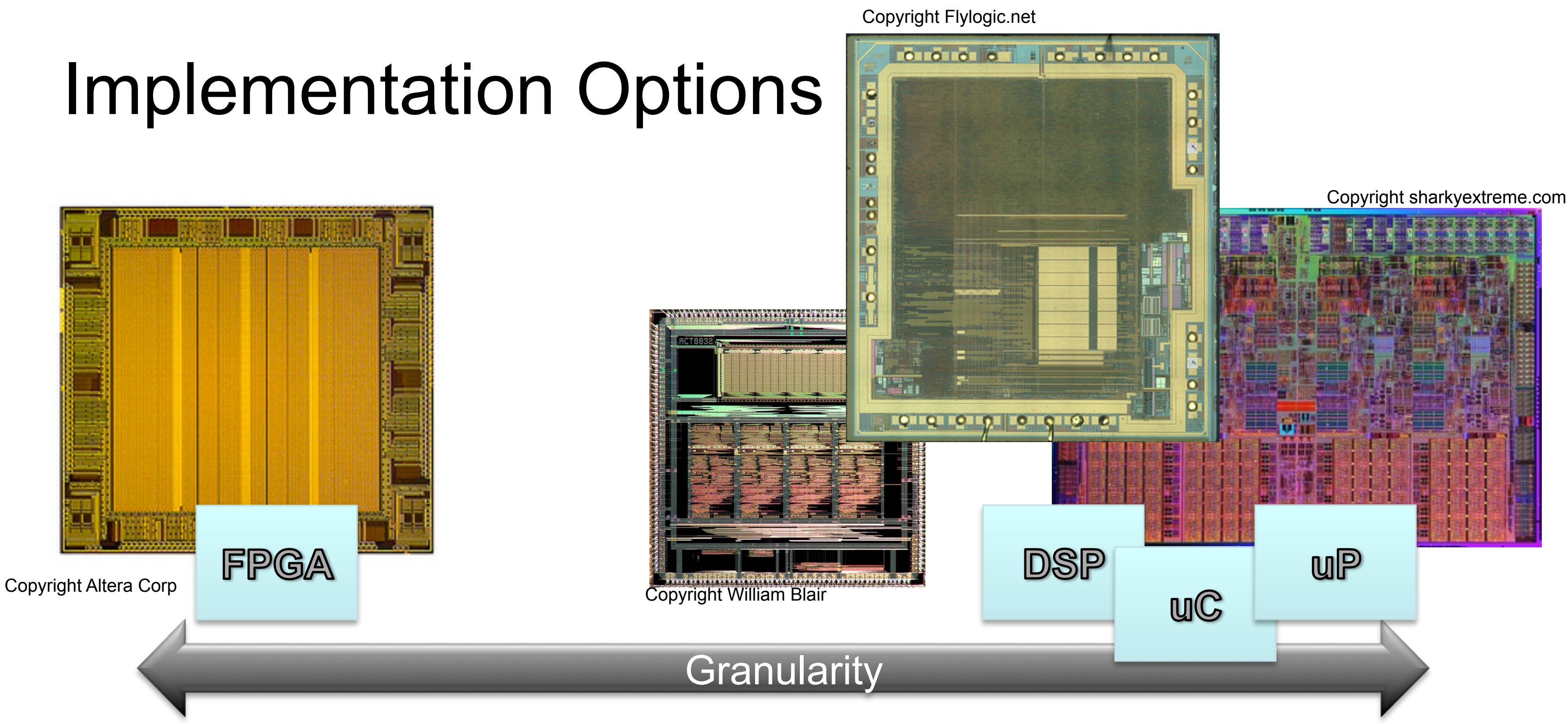
This is an important concept: It's the only justification that exists for designing RT systems using non-RT technologies:

Windows, Linux, USB, TCP-IP, WLAN

…

Real Time Logic

Implementation Options



Microprocessors

In order to get the best possible performance, microprocessors may make extensive use of features like

- Branch Prediction
- Caching
- Frequency Scaling
- Pipelining

All of these, and more, will effect the real time response of the processor.

If there's only one 'thread' of execution then many of the following effects are greatly reduced. We will define a thread of execution simply as a piece of code executed asynchronously with respect to another. In this way, threads include interrupt and exception handlers so single-threaded execution is actually very rare, even on the smallest of processors.

Microprocessors

In order to get the best possible performance, microprocessors may make extensive use of features like

- Branch
- Caching
- Freque
- Pipelini

All of these the processor.

If there's o llowing effects are greatly reduced. We will define a thread of execution simply as a piece of code executed asynchronously with respect to another. In this way, threads include interrupt and exception handlers so single-threaded execution is actually very rare, even on the smallest of processors.

Note that a processor is a collection of logic gates only, and is inherently deterministic to known influences. The source of non-determinism in the real time response of microprocessors is the way that features couple response to unknown influences

Microprocessors

Pipelining

Pipelining introduces dependencies between successive instructions (which may be from different threads of execution). This is not a problem if every instruction takes the same amount of time to execute, but if one instruction stalls, then those behind it stall as well.

A stall could be due to a floating point computation (execution time dependent on the actual data value), branch misprediction, cache miss or a number of other inter-instruction dependencies.

Microprocessors

Branch Prediction

When ever the processor encounters an ‘if’ construct (including things like ‘stop looping *if’*), it attempts to guess the answer and starts executing one branch before the outcome is actually known. This speeds up the case where the processor guesses correctly, but if the processor guesses incorrectly there’s a some amount of time (and energy) spent doing the wrong thing.

Branch prediction is generally deterministic if you look hard enough, but can couple different subsystems in unexpected ways.

For example, the run time of Thread 1 can be affected by the way Thread 2 has conditioned the branch predictor.

Microprocessors

Caching

The processor keeps a ‘cache’ of commonly used (in practice, usually *recently used*) data somewhere it can get quick access.

Like the previous items, this introduces timing dependencies between seemingly unrelated pieces of code. For example, being interrupted for a short time by a thread that uses lots of data may be more harmful than being interrupted for a long time by a thread that doesn’t touch much data (and therefore many cache lines).

It’s often stated that interrupts must be fast in order to have minimal impact on other processes. More generally, they should be deterministic in their execution time. Rarely would you find an interrupt specified in terms of its data access patterns *but you should!*

Microprocessors

Frequency Scaling

The previous items could, theoretically, be perfectly modeled. If all the running tasks are deterministic then there’s no problem; the problem they present is that it becomes hard to analyze the interaction between tasks, and to decouple real time and non-RT tasks.

Frequency Scaling cannot be modeled based on knowledge of the processor and software package alone. As the name suggests, this feature changes the speed of execution based upon the physical parameters of the processor. If the processor can be clocked faster without physical damage, it is.

Microprocessors

Frequency Scaling

This sounds great, but couples your Real Time system's performance to:

- Fan speed
- Dust
- Power supply stability
- Ambient Temperature
  - Meteorology
    - Time of Year
      - Celestial Progression
        - …

Microprocessors

Back to Determinism

So where do we stand on microprocessors versus determinism? High performance processors get that way by sacrificing plenty of things – determinism among them

In fact, modern processors are so non-deterministic, they are actually a useful source of cryptographic entropy. See for example the HAVEGE Project [1]. Note that worst-case execution time is usually still bounded, just not deterministic.

[1] http://www.irisa.fr/caps/projects/hipsor/

FPGAs

How do FPGAs compare to microcontrollers in terms of determinism and real time performance?

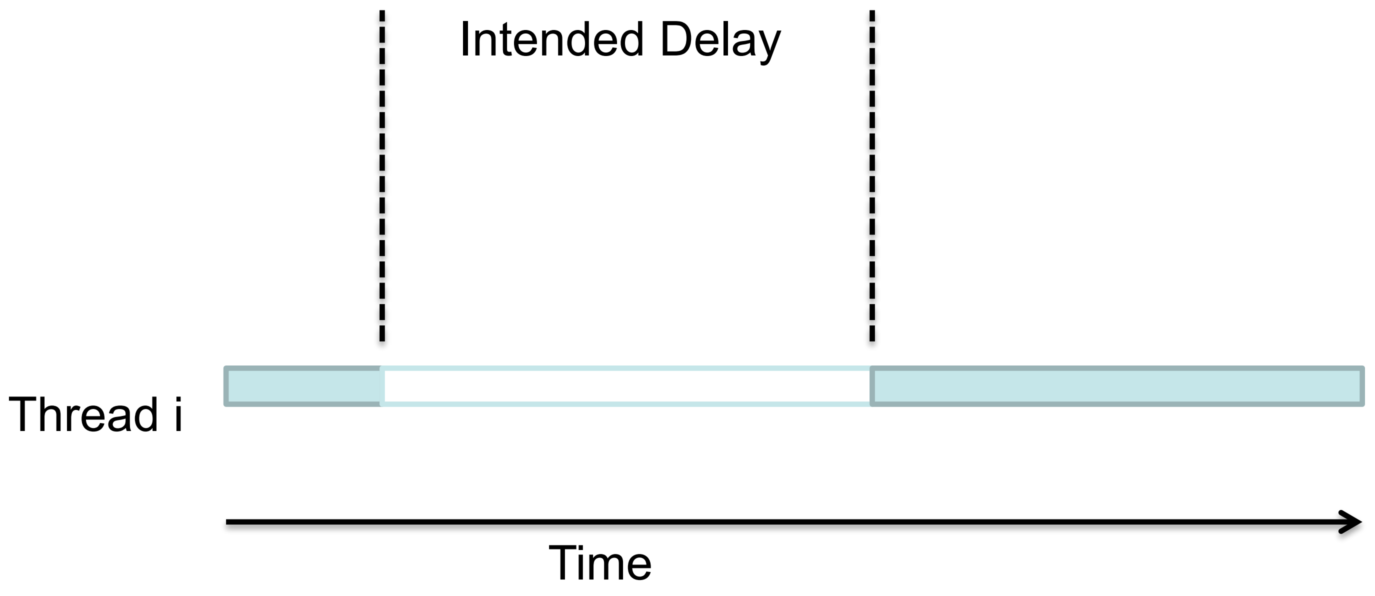
FPGAs do have an intrinsic advantage over microprocessors in terms of determinism as unrelated tasks are performed in physically separate places so there is not necessarily any conflict for resources nor coupling between unrelated pieces of code.

If your FPGA design includes some shared resource and an associated arbitration algorithm (memory, radio modem etc.) then you start introducing couplings and associated non-determinism.

These interconnections are very explicit though, and relatively easy to audit.

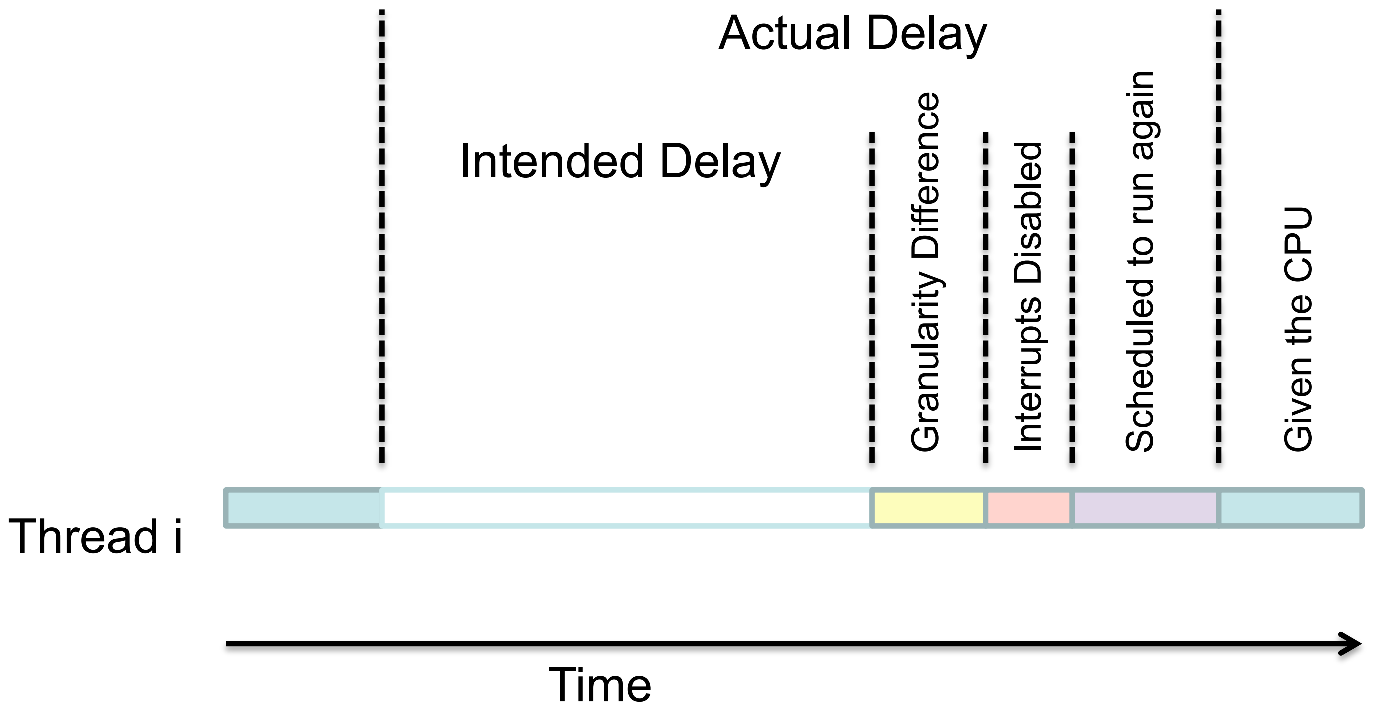
Interacting with Time

Delaying execution for a fixed amount of time is a commonly-used alternative to busy-waiting or interrupts



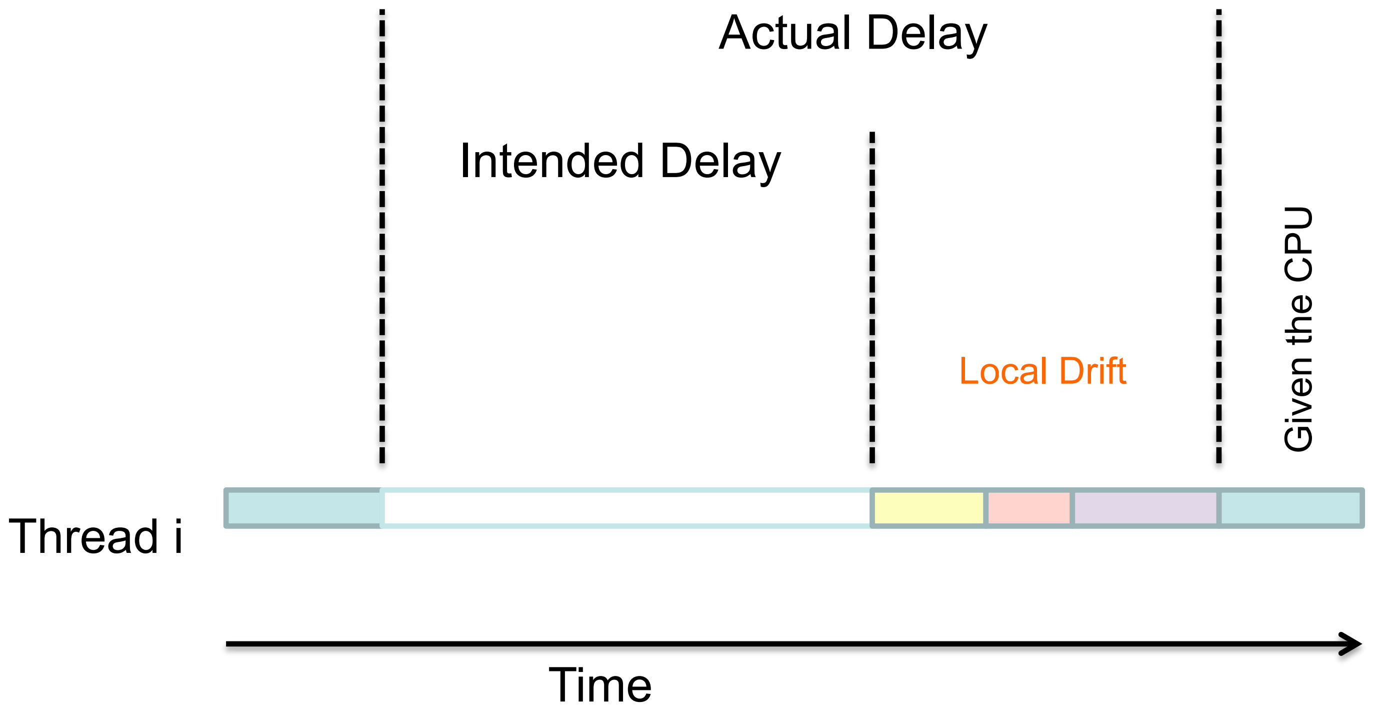
Interacting with Time

But the semantics of primitive delays aren't as one might expect. They're precise only in their *lower bound.* In English, rather than "Stop the task then run again after n", "Stop the task, then after the CPU's idea of n, make it eligible to be run again".



Interacting with Time

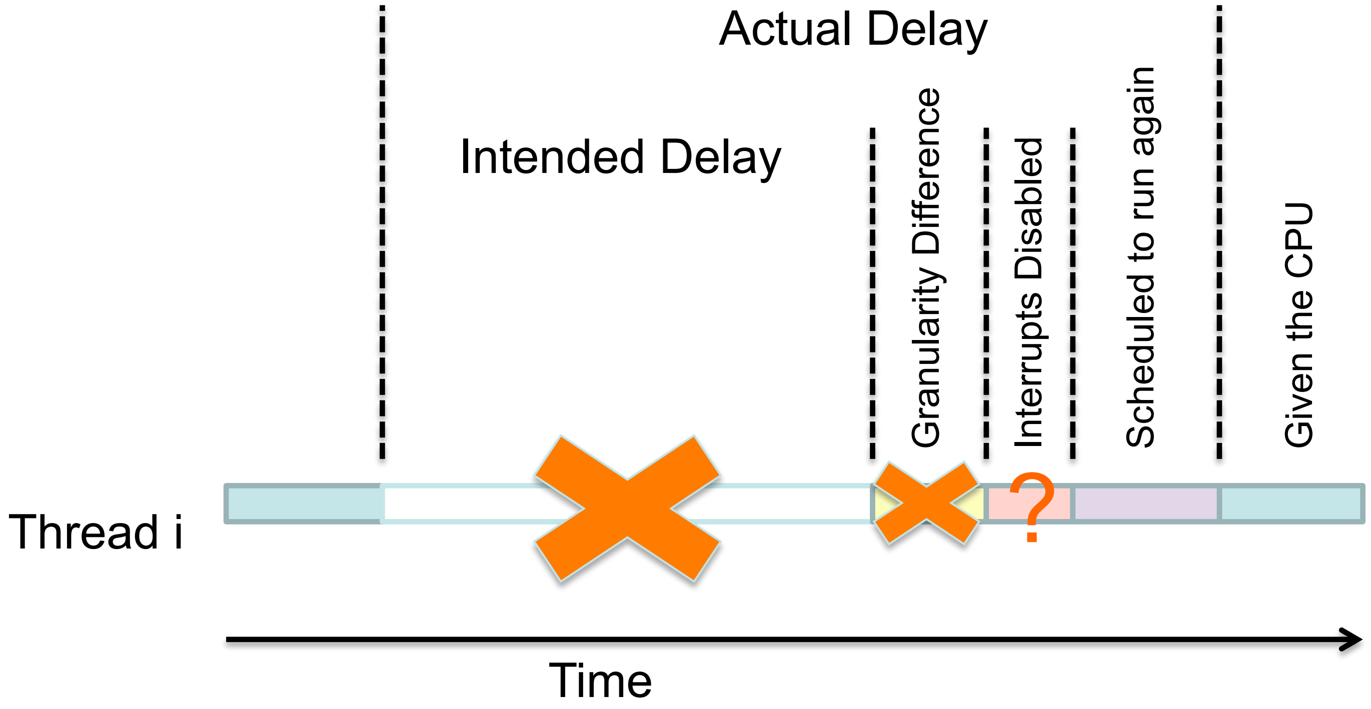
This difference is termed Local Drift



Interacting with Time

What is the effect of a ‘zero’ delay?

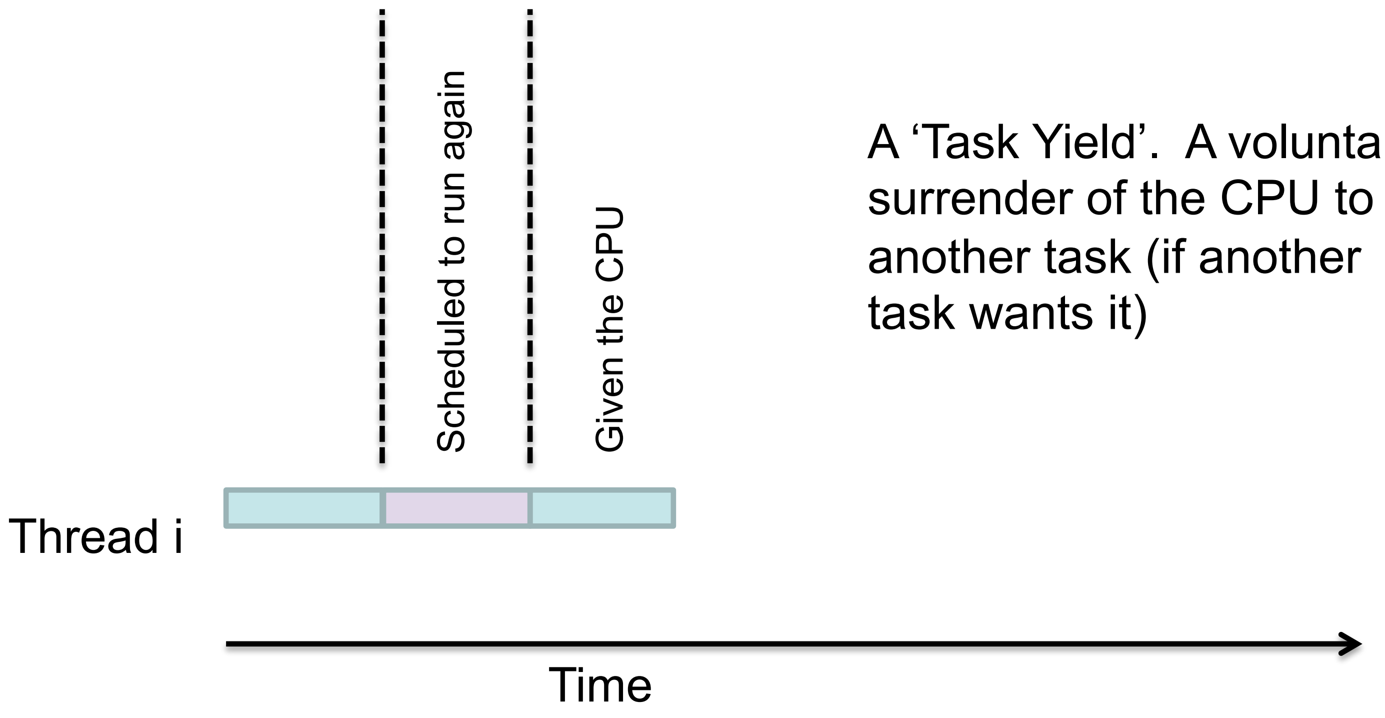
“Stop the task, then after the CPU’s idea of 0, make it eligible to be run again”.



Interacting with Time

Returning to our description of delays, what is the effect of a 'zero' delay?

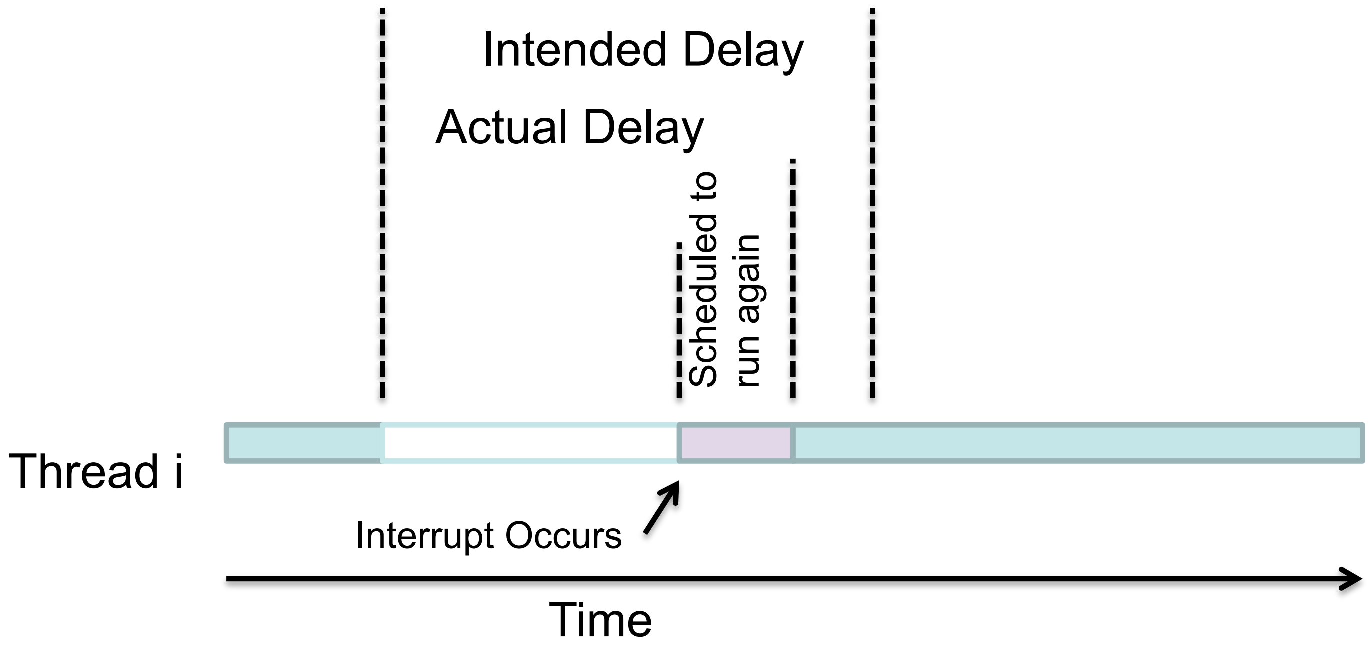
"Stop the task, then after the CPU's idea of 0, make it eligible to be run again".



A 'Task Yield'. A voluntary surrender of the CPU to another task (if another task wants it)

Interacting with Time

If the thread is enabled for interrupts then it might be made runnable again sooner: “Stop the task running, then after the CPU’s idea of n, make it eligible to be run again unless someone(thing) wants its attention sooner, in which case run it at that time instead”



Interacting with Time

Naïve delay attempt in C (assumes library support).

This suffers from local drift but also cumulative delay due to the finite time taken to execute the action and to loop around.

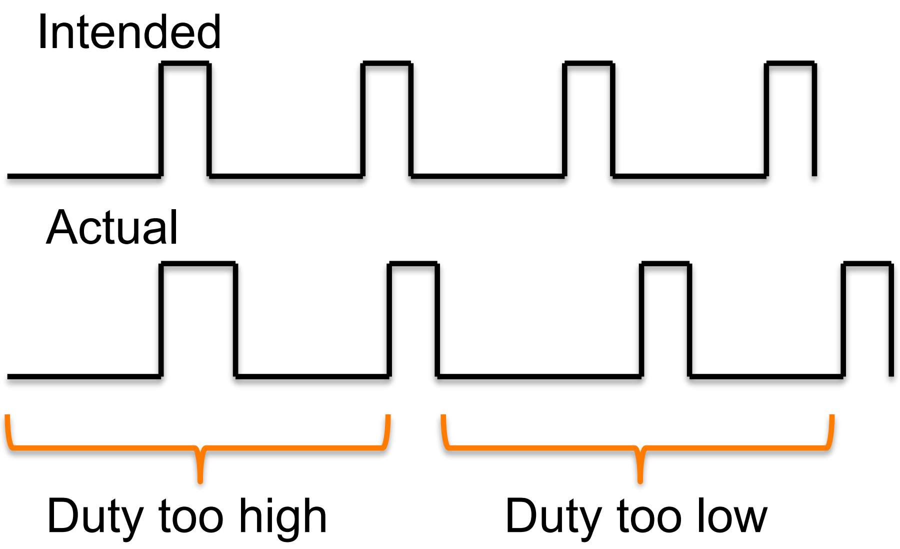
```
while(1) {
    do_action();
    _delay_ms(Interval);
}
```

Interacting with Time

Naïve PWM attempt in C

```
while(1) {
    pin_high();
    _delay_ms(Interval * Duty);
    pin_low();
    _delay_ms(Interval * (1 – Duty));
}
```

Despite being naïve, it's actually a very common construct in very simple single-threaded microprocessors where the possible values of local drift are very small or zero. In the general case, this may actually be physically dangerous.



Interacting with Time

```
while(1) {
    start = get_time();
    do_action();
    _delay_ms(Interval – (get_time() – start));
}
```

Attempt to remove cumulative delay by timing the action in the middle and subtracting this number from the overall delay time.

Delay time calculation is not atomic. If an interrupt arrives at the wrong time (including the timer interrupt that's probably driving the clock behind the `get_time()` function) then the delay can still be significantly wrong. Don't do this unless you've taken separate measures to ensure the atomicity of that statement!

Interacting with Time

```
next_time = Clock + Interval;
begin loop
   Do_Action;
   delay until next_time;
   next_time = next_time + Interval;
end loop
```

The only correct way to do these kinds of delays is to use absolute delay constructs, rather than the relative delays we've seen up until now. These will delay on average for the right time, even if the action is sporadically longer than that. Neither C nor the standard C libraries support absolute delays so the example above has been written in Ada. If you are required to do proper absolute delays in C then it is possible to write your own absolute delay in that language.

Interacting with Time

Constructs for specifying timing constraints are strictly part of a programming language and/or its support libraries.

Relative delays only
C, generally simple languages never designed for RT tasks. In C (POSIX), absolute delays may be constructed out of timers and signals.

Both Relative and Absolute delays
Ada, Real Time Java, most process-oriented languages

Absolute delays only
Ada Ravenscar Profile, Occam2, most languages built explicitly for real time use

Safety and Liveness

If a program has been written to interface with real time, the following questions can be asked:

- Is the right thing going to happen (liveness)
- Is the wrong thing not going to happen (safety)

The property of liveness is that of progress; if a program has to wait for an event (resource), will that event (resource) eventually come? This is a key concept when scheduling resources that have to be used by several threads (including the CPU itself) and will be revisited during discussions of operating systems.

Safety is less subtle. Simply, do all failure modes lead to something safe happening over something bad (controlled shutdown, self destruct…).