EEE3096S: Embedded Systems II

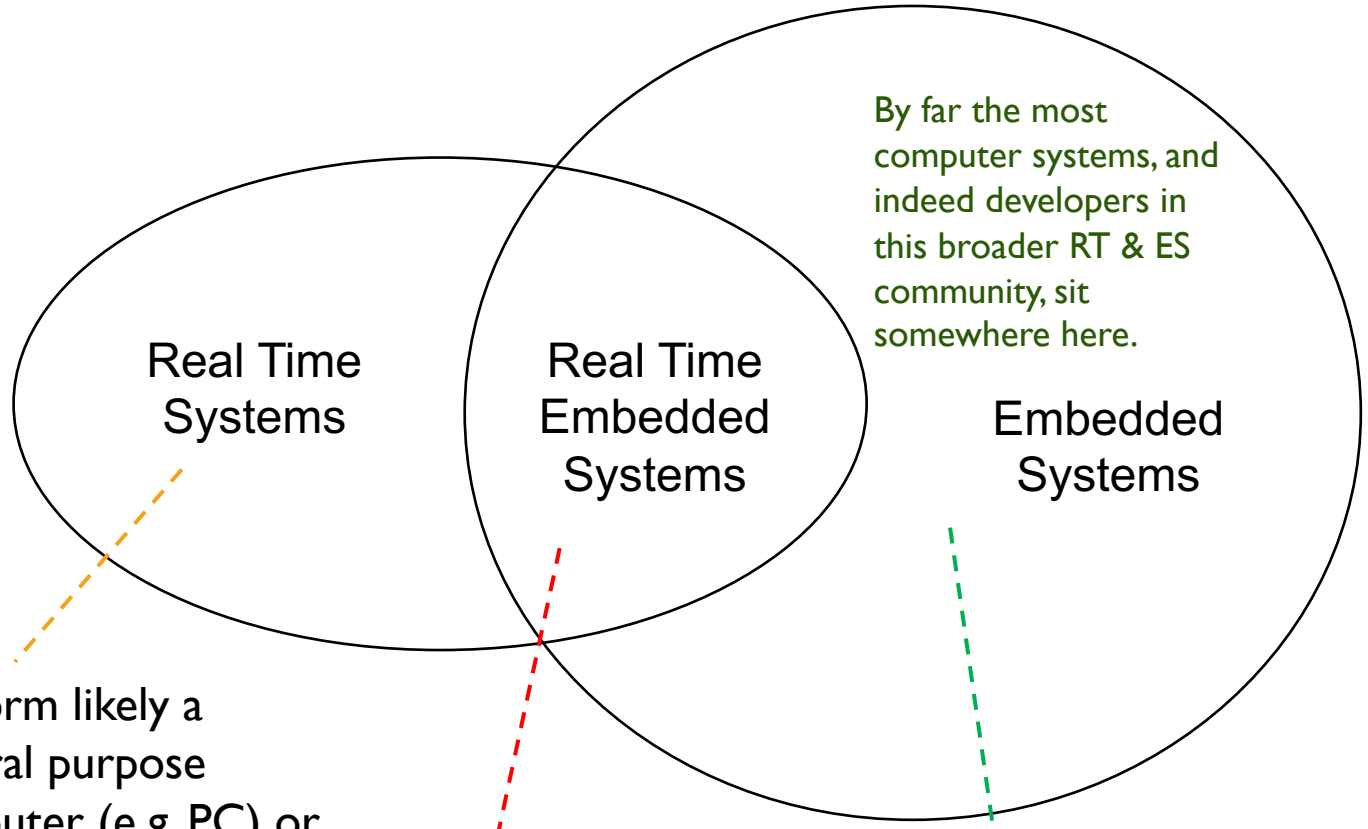
LECTURE 2:

REAL TIME SYSTEMS
HARD AND SOFT DEADLINES

Presented by:

Dr Yaaseen Martin

Electrical Engineering
University of Cape Town



Platform likely a general purpose computer (e.g. PC) or resides in the cloud, not much choice on hardware design.

Can be a challenge to tame these beast (by 'tame' I mean ensure hard deadlines are ensured).

A special-purpose computer built into a larger system (you know what these are already!)

EXAMPLES

- **Real Time Embedded Systems (RTES):**
  - Nuclear reactor control
  - Flight control
  - GPS
  - MP3 player

- **Real Time Systems (RTS)**:
  - Stock trading system
  - Skype
  - Pandora

- **Embedded Systems (ES)**:
  - Home temperature control
  - Sprinkler system
  - Washing machine, refrigerator, etc.
  - Blood pressure meter

CHARACTERISTICS OF REAL-TIME SYSTEMS

The main factors of an RT system are:

- Event-driven
  - The system is reactive to inputs or events (i.e. external stimuli)
- High cost of failure
  - e.g. damage to expense equipment or environment or even loss of life.
- Concurrency/multiprogramming
  - Needing to do multiple things at once or actions that occur very close together in time
- Stand-alone/continuous operation
  - Often needs to run without depending on operation of other devices, needs to run for long periods without fail.
- Reliability/fault-tolerance requirements
  - Needs to keep running consistently for long periods, and often needs to keep running even in the case of faults or failing components (at least to some extent – e.g. graceful degradation or 'fail soft')
- Predictable behavior (this is usually the most important aspect)
  - Clear understanding of how and *when* system reacts in any given situation

RT RELATED TERMINOLOGY

- **Hard real-time:** systems for which it is absolutely imperative that responses occur within the required deadline. (e.g. flight control systems).
- **Soft real-time:** systems for which deadlines are important but which will still function correctly if deadlines are occasionally missed (e.g. a weather data logging system).
- **Real real-time:** systems that have at least one hard real-time system and for which the response times are usually very short. e.g. Missile guidance system.
- **Firm real-time:** systems that can survive task failures as long as they are adequately spaced, i.e. *infrequently* missed deadlines

In reality, a system is more likely a combination of some of these cases; more accurately viewed as a cost function being associated with missing each deadline.

SOFT VS HARD REALTIME

- Soft Realtime
  - More slack in the implementation
  - Timings can be suboptimal without being incorrect
  - Problem formulation (e.g. working our dependencies and scheduling delays) can be much more complicated than hard real-time
  - Approaches tend to be either:
    - Set somewhat loose hard timing constraints
    - Do design and testing (more informal testing as opposed to 'formal verification methods'*)
    - Formulate as an **task optimisation** problem, not **timing optimisation**

* If you haven't encountered this term before, please read: https://en.wikipedia.org/wiki/Formal_verification

SOFT VS HARD REALTIME

- Hard Realtime
  - Much stricter implementation
  - Timings need to be precisely worked out (formal verification more commonly used)
  - Problem formulation is more precise and focused, less software ‘clutter’ of soft RT
  - A potentially clearer and simpler problem formulation but deadlines may still be difficult to satisfy and/or prove.
  - Usually about ensuring hard deadlines are satisfied* (which have a high cost or catastrophe if not handled in time)

*However, based on the literature, it’s not totally clear that a hard realtime system must have hard deadlines, but it abundantly indicates a stricter and likely more difficult and thorough design.

STATIC VS DYNAMIC RT SYSTEMS

when something needs to be responded to

- Static:
  - Task arrival times are finely predicted
  - Allows static analysis (i.e. can be fully analysed offline at compile-time or before runtime)
  - Can design around well-balanced resource usage (i.e., minimize idle time, optimize system processing costs)
- Dynamic:
  - Arrival times of tasks may be unpredictable
  - Static (compile-time) analysis often not possible or very difficult (or too costly) to model accurately.
  - Processor utilization can be very varied, likely leading to over-design and excessive processing resources.
  - Needing to avoid over-simplifying assumptions (e.g., assuming all tasks are independent, could overlap, which could in practice be very unlikely).
  - Often makes for more difficult designs.

MULTI-PROCESSOR OR DISTRIBUTED SYSTEM: COMMON APPROACH TO HARD REAL-TIME SYSTEMS

- Typically a Hard RT system has only a few **hard deadlines** and lots of **soft deadlines**.
- Nowadays computing resources are cheaper than they used to be, with more features … such as convenient processor SoCs packaged, e.g. FPGAs + Microcontrollers.
- Thus, in effect, delivering a 'Hard RTES' system is likely going to involve a design that has more than one processing resource – the design is typically separated into two, i.e. a part for Hard-RT and a part for Soft-RT.
- This is technically a multiprocessor system or even distributed system (separate computers working collaboratively to satisfy requirements)

* The EEE4120F (High Performance Embedded Systems) course dives into these issues more deeply.

PERIODIC/SYNCHRONOUS DESIGN

NB: This is about overall processing regularity of the system.

- **Periodic** = each task (or task group) executes repeatedly at specific period
- Well-suited to hard RT and simplifies static analysis.
- Matches character of many real problems.
- Could have tasks with deadlines that are: smaller, equal, or greater than their period.
- Single- or Multi-periodic rate
  - **Single rate**: One period in the system, simple but inflexible (e.g. often used in wireless sensor nets)
  - **Multi rate**: Multiple periods; but usually simplified as harmonics to simplify the design

Note: don't assume this implies system inputs will be polled, may be a dangerous assumption as the system may need near simultaneous inputs or rapid input reads but ensuring hard deadline met likely at a higher level (some inputs may be asynchronous to satisfying deadlines).

APERIODIC/ASYNCHRONOUS DESIGN

- An **aperiodic**, or 'sporadic', system is designed around being asynchronous or reactive.
- Creates a dynamic situation (periods between events/responses may become very varied)
- Bounded arrival time interval easier to handle
- Unbounded arrival time intervals are impossible to handle with resource-constrained systems (i.e. think rapid series of inputs).

Note: don't assume this implies the system inputs will be polled, this may be a dangerous assumption as the system may need to support near simultaneous inputs or rapid input reads but the higher level hard deadline may be at a higher level, some inputs possibly asynchronous to operations to satisfy deadlines.

HARD VS SOFT REALTIME SYSTEM CHARACTERISTICS

Characteristic	Hard RTS	Soft RTS
Response time	Hard required	Soft required
Peak load performance	Predictable	Degradable
Control of pace	Environment	Computer
Safety	Often critical*	Non-critical
Size of data files	Usually small/medium	Can be large
Redundancy type	Active (concurrent)	Check-point recovery (e.g. can rollback and fix things)
Data integrity	Short-term	Long-term
Error detection	Autonomous and usually rapid in situ (can't rely on human intervention)	Potentially user-assisted

* The concept of 'safety-critical systems' are particularly pertinent here:

Definition of **Safety-Critical System (SCS)**: or 'life-critical system' is a system whose failure could result in one (or more) of the following outcomes: death or serious injury to people, or loss or severe damage to equipment/property/environments.

COMPUTING RESOURCE SCARCITY

- Scarcity of computing resources is a characteristic that often impacts ES and RTES designs.
- Resource availability also characterizes an RTES design:
  - *Abundance of resources*, i.e. all computing and timing constraints easily met → greatly simplifies design
  - *Totally inadequate resources (TIS)*, i.e. major shortfall of resources → either simplifies design (negotiate alternate solution with client) or ask for more resources
  - *Sufficient but scarce resources* → this makes for harder… and more interesting… problems, battling optimisation, puzzling out how to schedule tasks while satisfying deadlines etc. Tends to be where RTES development heads, largely due to wanting to minimize cost and maximize features and performance.

INTERNET OF THINGS (IOT)

- Main insights are:
  - IoT = the billions of physical embedded devices around the world that are connected to the internet, all collecting and delivering data.
  - This is possible from super cheap computer chips and ubiquity of wireless networks… together making it possible to turn (almost) anything into a part of the IoT.
  - The massive connectivity of these sensors and other IoT devices brings digital intelligence to devices that would be otherwise be (independently) dumb, enabling them to communicate real-time data without needing implicit involvement of humans.
  - IoT is making the fabric of the world around us smarter and more responsive, merging the digital and physical universes.

INDUSTRIAL IOT (IIOT)

- IIoT can be considered the extension or specialsation of IoT devices towards industry application.
- The main focus of IIoT is:
  - Machine-to-Machine (M2M) communication
  - Big data
  - Machine learning (to optimise industrial processes)
- IIoT is predominantly about enabling industries (or more generally businesses) to have better monitoring, efficiency, control and reliability in their operations.
- The IIoT encompasses many industrial applications, including: robotics, software-defined production processes, logistics, medical devices, among others.

ES DESIGN OPTIMISATION METRICS

- Optimisation of an ES, especially a RTES, can be complex and involves a potential multitude of conflicting dimensions (i.e. features may work against each other) e.g.:
  - more memory → higher cost
  - less power → slower operation
- Usually, and what ES engineers often spend much time discussing, is the ‘optimisation metrics’ they are using or their optimisation schedule (if not ‘optimisation matrix’ that is used to work out satisfactory compromises).

ES OPTIMISATION DESIGN METRICS

Design Metric: a measurable feature of an embedded system's design, e.g. its performance, cost, time for implementation, safety, etc.

As mentioned, these are typically conflicting requirements i.e. optimising one doesn't necessary optimise the other – indeed may instead reduce it.

As you have probably now realized, with your existing understanding of engineering, there could be masses of such optimisation metrics….

But the 'usual suspects' are the following (not in priority order) :

1. NRE cost (nonrecurring engineering cost)
2. Unit cost
3. Size
4. Performance
5. Power Consumption
6. Flexibility
7. Time-to-prototype
8. Time-to-market
9. Maintainability
10. Correctness