

---

---

# EECE 276

# Embedded Systems

Requirements:  
Problems, types, real-time

# Requirement Engineering

---

- Capturing and documenting what the user wants.
- Stages:
  - » Preliminary study -> Feasibility report
  - » Elicitation -> Domain model (interviews)
  - » Definition -> Definition documents (writing it up)
  - » Validation -> Specification documents (checking it)
- Iterations:
  - Elicitation <-> Definition <-> Validation
- Correctness is essential (hard to fix later)

# Requirement Types

---

1. Functional (what it does)
2. External interfaces (how it connects)
3. Performance (what is expected)
4. Logical database (types of info used)
5. Design constraints (standards, etc.)
6. Attributes (reliability, availability, security, maintainability, portability)

2-6: *Non-functional* Requirements

# Requirements for real-time systems

---

- Typical examples
  - » Timing:
    - WCRT for every stimulus/response pair
    - Sustained data rates
    - Maximum latency
  - » “Graceful degradation”
  - » Availability, reliability, safety
  - » Power consumption
  - » ...
- To capture:
  - » Formal “requirement” languages

# Formal Requirement Languages

---

- “Formal methods” – mathematical, abstract
- Why?
  - » Consistency checking (Is it consistent?)
  - » Model checking (Is a state reachable?)
  - » Theorem proving (It is always true that ...)
- Example:
  1. If interrupt A arrives, the task B stops executing.
  2. Task A begins executing upon arrival of interrupt A.
  3. Either task A is executing and task B is not, or task B is executing and task A is not, or both are not executing.

“interrupt A arrives” :  $p$

“task B is executing”:  $q$

“task A is executing”:  $r$

1.  $p$  implies  $q$
2.  $p$  implies  $r$
3.  $(r \text{ and not } q) \text{ or } (q \text{ and not } r) \text{ or } (\text{not } r \text{ and not } q)$

# Consistency checking

---

- Are these consistent?

1.  $p$  implies  $q$
2.  $p$  implies  $r$
3.  $(r \text{ and not } q) \text{ or } (q \text{ and not } r)$   
or  $(\text{not } r \text{ and not } q)$

- Check: build a truth-table with all the possible values for  $p, q, r$ , and see if 1., 2., and 3. are true simultaneously.
- Example: TACAS consistency checking
  - Terminal Area Collision Avoidance System by FAA

# Limitations of formal methods

---

- Can support verification but not validation
  - » Verification: “the system is built right” (e.g. consistent) – w.r.t. ‘requirements’
  - » Validation: “the right system is built” (e.g. what the customer expects) – w.r.t. expectations
- We are verifying formulas but NOT the actual system!
  - » We still need testing (which can never be complete...)
- Difficulty in use...formal reasoning