Large Project Design
Large Project Design

1. Identify Modules
2. Clearly define modules
   – Define Interfaces
   – Allocate Resources
   – DIAGRAMS!!!
3. Check for conflicts:
   – Implicit (e.g. 2 modules using 1 TOC)
   – Potential (e.g. If Serial ISR takes too long to run, other ISRs will have issues)
Large Project Design

4. Define (& Implement) Tests
5. Implement modules in isolation
   – Can happen in parallel
   – TEST AS YOU GO!!
6. Cross-Validate (Test/Verify your partner’s work)
7. Integrate (SLOWLY!!)
   – Test-as-you-integrate

• Don’t forget your design cycle!
Top-Down Design

- Divide and conquer:
  - Start with a set of high level tasks (that will be called from `main()`)
  - Recursively splitting each task into subtasks until a level of complexity is attained that permits the definition of reasonably simple function modules.
Bottom Up Design

• Identify various components that will be required by the program
• Build them (or assemble them from libraries) in isolation from any design of the overall program structure.
• Assumes the appropriate argument types are known!
• Function interfaces can be designed, after which the internal implementations are usually straightforward.
• Advantage: more generic codes/modules
• Disadvantage: No clear indication of how the individual program elements should be merged together
Large Project Design

• In reality, use both.
• Top-Down works well when initially breaking up the project.
  – Logical breakdown of initial functional spec
• Bottom-Up design works well when the capabilities of the hardware are well known and defined.
  – Each module has some hardware or maths at it’s heart
• *Be careful that they properly meet in the middle!*
• *Don’t restrict yourself to sequential code!*
Module Testing

• Module tests should be written BEFORE the code is written

• Reference to module spec

• DO NOT CHANGE TEST ROUTINES!!!

• Basic test routine – write to LEDs
Integration

• If design and module tests done well, allow \( \frac{1}{4} \) to \( \frac{1}{3} \) project time for integration/debugging.
  
  – \( \frac{1}{3} \) Design
  – \( \frac{1}{3} \) Module Implementation
  – \( \frac{1}{3} \) Integration

• If done poorly, allow \( \frac{2}{3} \) to \( \frac{3}{4} \) project time.
Final Validation

• If testing is done on all modules and stages of integration, this is only a formality.....

• As with all other tests, think about these early in the design stage