Real Time System Testing (MIT 16.070 Lecture 31)

MIT 16.070 Real Time System Testing

- The next three lectures will focus on:
  - Lecture 30: *(R 11.3)*
    - How to minimize failure in real time systems
    - Methods used to test real time systems
  - **Lecture 31:** *(R 13)*
    - **What is Software Integration?**
    - **Test Tools**
    - **An example approach for integration and test of the MIT 16.070 final project**
  - Lecture 32: *(R 11.4)*
    - Fault Tolerance
    - Exception Handling
    - Formal Test Documentation

Hardware / Software Integration

What is Hardware/Software Integration?

- Uniting modules from multiple sources to form the overall system
- Verifying the system works
  - according to response time constraints (overall timing)
  - according to the system requirements

Think of it as building a jig-saw puzzle one piece at a time
or cooking a meal from a recipe.

Hardware / Software Integration

<u>System Unification - The Steps Involved</u>

1. Establish a baseline: Identify the different components and/or modules
2. Carefully select your test tools - what can make your job easier?
3. Determine the best way to fit them together into larger pieces
   - Link the modules into one load module to form a piece of the system
     - *Likely to surface errors in unresolved external symbols, memory assignment violations, page link errors, etc.*
     - *Often it makes sense to build the cycling infrastructure (tasking) first.*
   - Test that piece of the system
   - Link in more modules until you have the entire system covered.
   - In systems running on multiple platforms, you will need at least one load module per platform and a plan for testing the system as a whole.

Hardware / Software Integration

System Unification - The Steps Involved (continued)

4. Regression Testing

   If a system test fails and the problem has been fixed, you must <u>rerun</u> any affected tests to make sure your fix hasn't broken something else:

   - All module-level test cases for any changed module
   - All system level test cases

5. Write up your results

Real Time Testing Tools

- **Test Tools - Low Level**
  - **Code Comparison (identifies differences between two versions of the same program)**
  - **Complexity Analyzer (uses statistical mathematics to measure code complexity)**
  - **Syntax Checkers (checks for correct parsing of input and handling of incorrectly formatted data)**
  - **Symbolic Debugger (runs a program and tracks variables and the order in which instructions are executed)**
  - **Data Flow Analyzer (ensures data used by the program has been properly defined)**
  - **Control Flow Analyzer (analyzes logic branches within a program)**
  - **Peer Review (uses peers to review programmer's code)**
  - **Walk-Through (asks the programmer to explain his or her code)**

Real Time Testing Tools

- **Test Tools - Intermediate Level**
  - **Emulator**
  - **Logic Analyzers**
  - **Bus Analyzers (PCI, 1553, RS232, VME, etc.)**
  - **Debugging tools to allow monitoring of events (context switches, interrupts, semaphores, message sending, etc.)**
  - **Tools to allow real time monitoring of variables**
  - **Telemetry gathering and post processing/analysis tools.**
  - **Console I/O**
  - **Debugger**
  - **Simulators to provide controled inputs to the system (GPS, autopilot data, etc.)**

Real Time Testing Tools

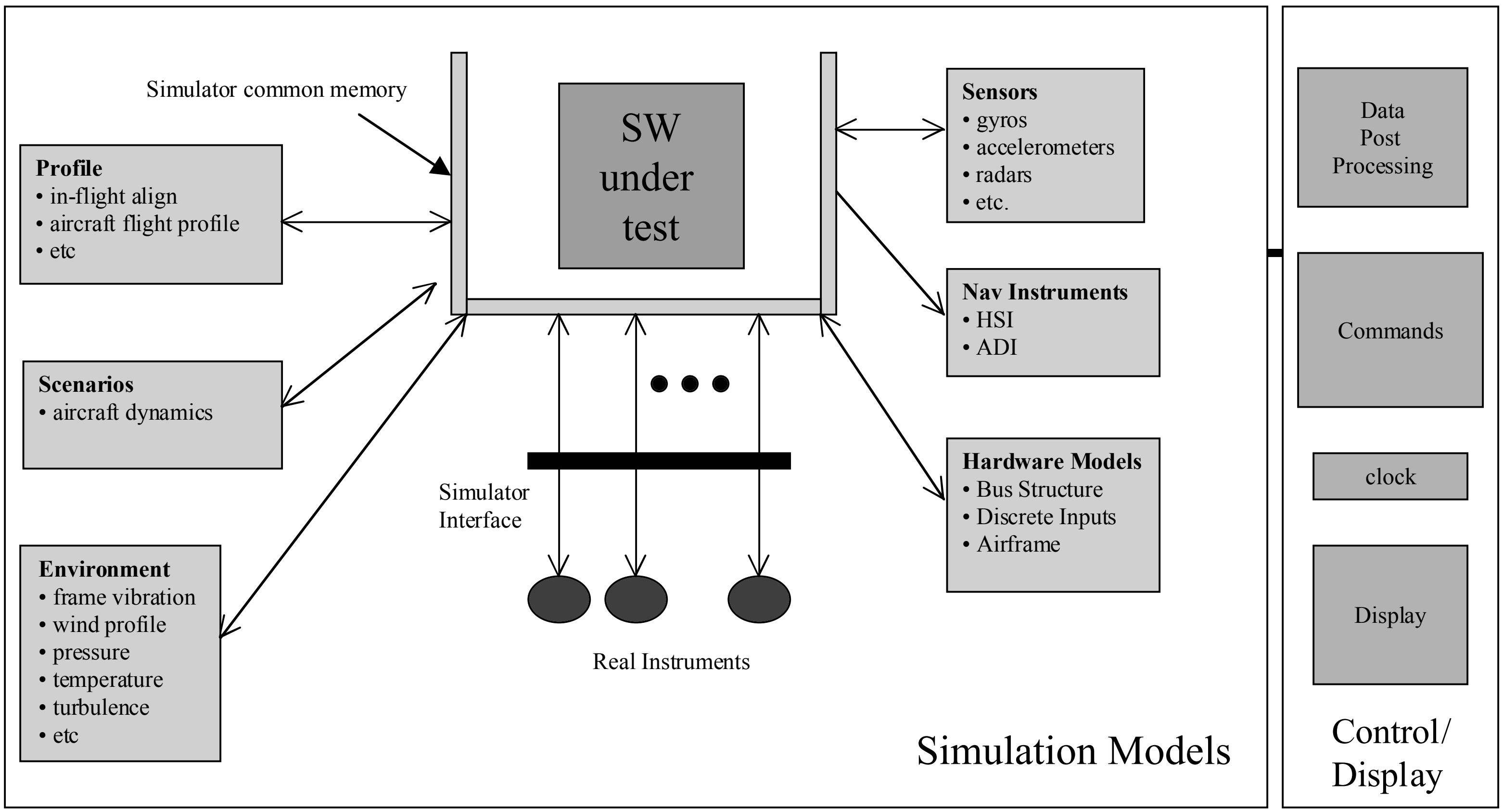
- **Test Tools - High Level**
  - **End-to-End closed-loop simulation (software that represents the whole system into which the embedded code will be placed)**
  - **Hardware-in-the-loop test bed (software and hardware to represent the whole system)**

Advantages of Using a Good Simulation

- **Uses real flight code**
- **Provides rigorous realistic dynamics**
- **Provides external conditions (including failure conditions, off-nominal cases, etc.)**

Real Time Testing Tools

- Test Tools - A Simulation Test Bed Example



Hardware/Software Integration - An Example

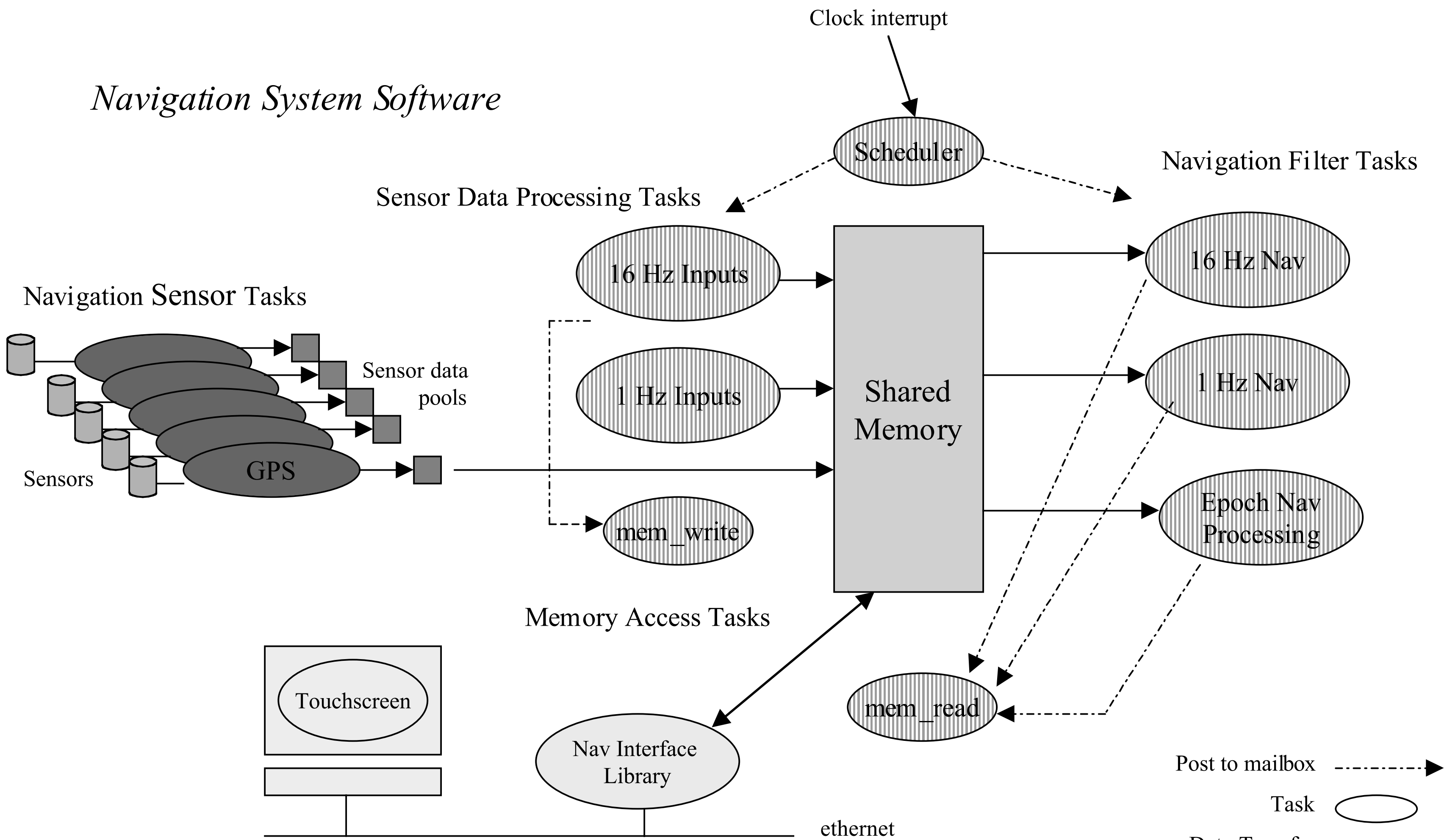
The Problem:

- Develop a system to provide underwater and surface navigation data to the crew of a submarine allowing a 1-sigma nav error of 10 meters

The Design Constraints

- Use existing navigation sensors for velocity, position and attitude data (includes various hardware interfaces)
- Create a user interface via touchscreen
- New navigation data is available at 16 Hz

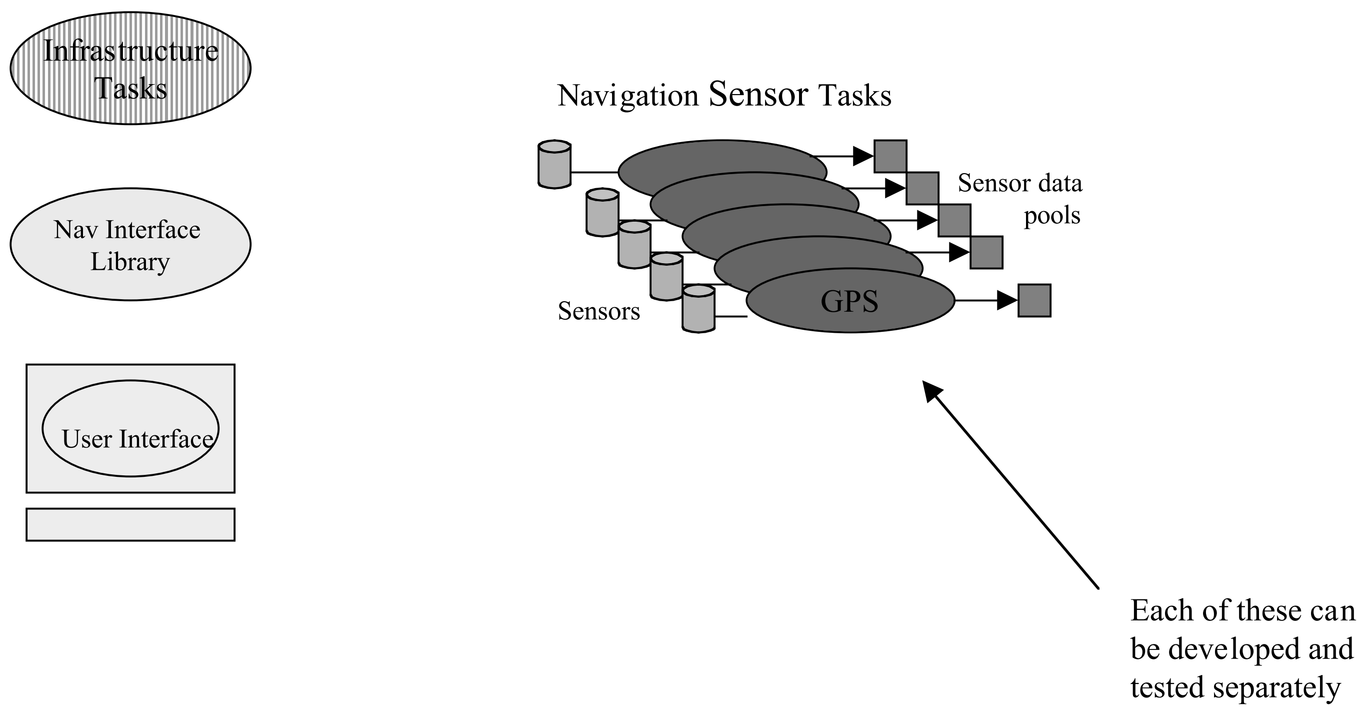
The Navigation System Software Architecture



Hardware / Software Integration - An Example

System Unification - The Steps Involved

1. Establish a baseline: Identify the different components and/or modules



Hardware / Software Integration - An Example

System Unification - The Steps Involved (continued)

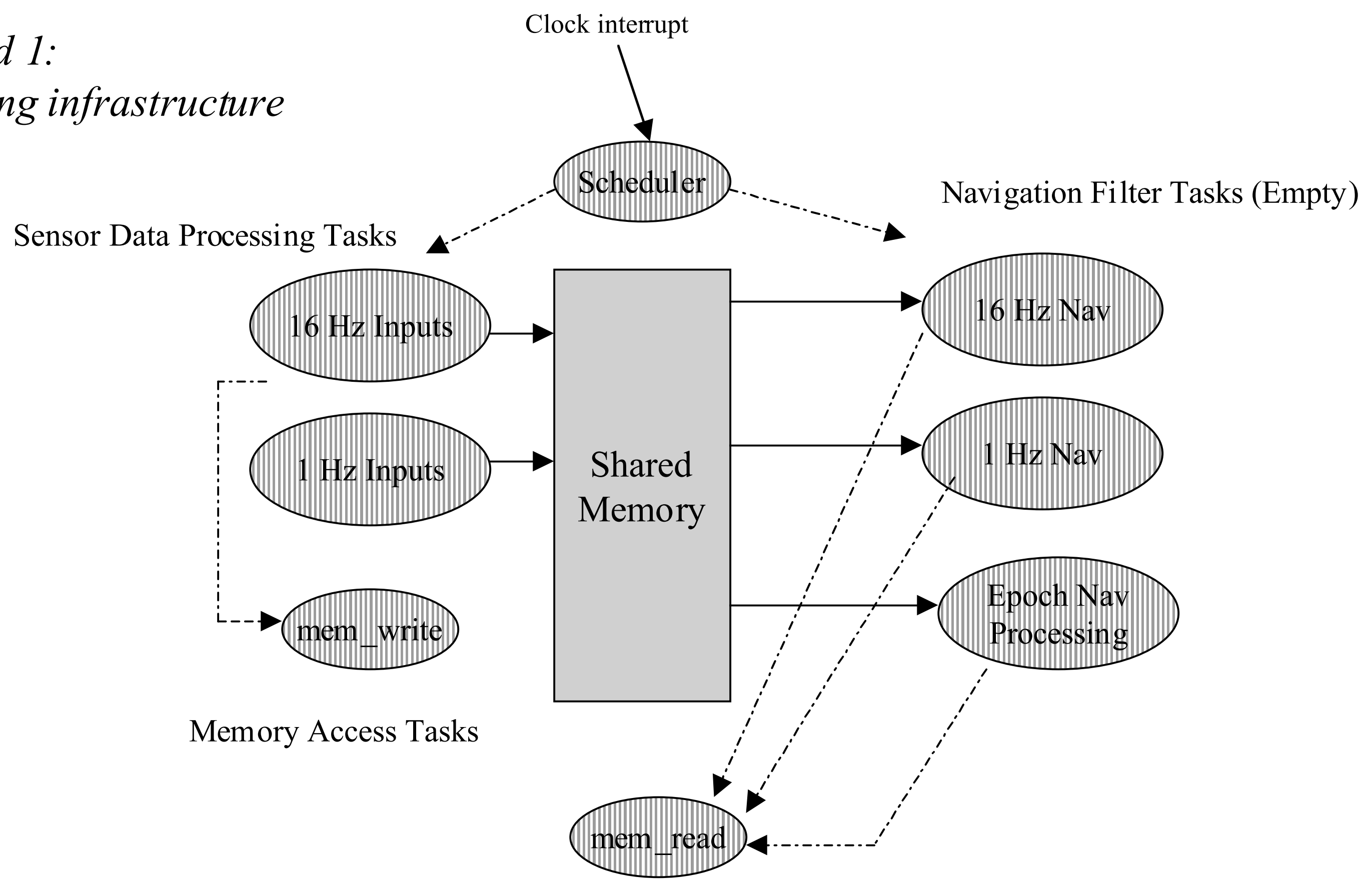
2. Carefully select your test tools - what can make your job easier?
   - Software simulation for each of the navigation sensors
     - Dynamic mode
     - Static mode
   - Real sensors when practical (GPS, Inertial Unit)
   - Hardware-simulated instruments (dials, buttons)
   - VAX Terminal for debug print
   - SUN Workstation running OS shell (window on real time operation)
   - VME Bus Analyzer
   - Peer Reviews / Design Reviews

Hardware / Software Integration - An Example

System Unification - The Steps Involved (continued)

3. Determine the best way to fit them together into larger pieces

Test Build 1:
The tasking infrastructure

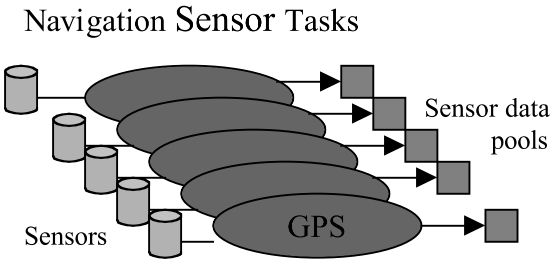


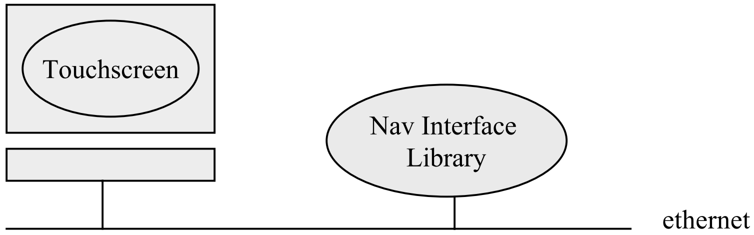
Hardware / Software Integration - An Example

<u>System Unification - The Steps Involved (continued)</u>

3. Determine the best way to fit them together into larger pieces

Unit Test & Software Integration Test the Subsystems
- Load module for each of the Sensor Tasks
- Load modules for the User Interface (one for each processor)



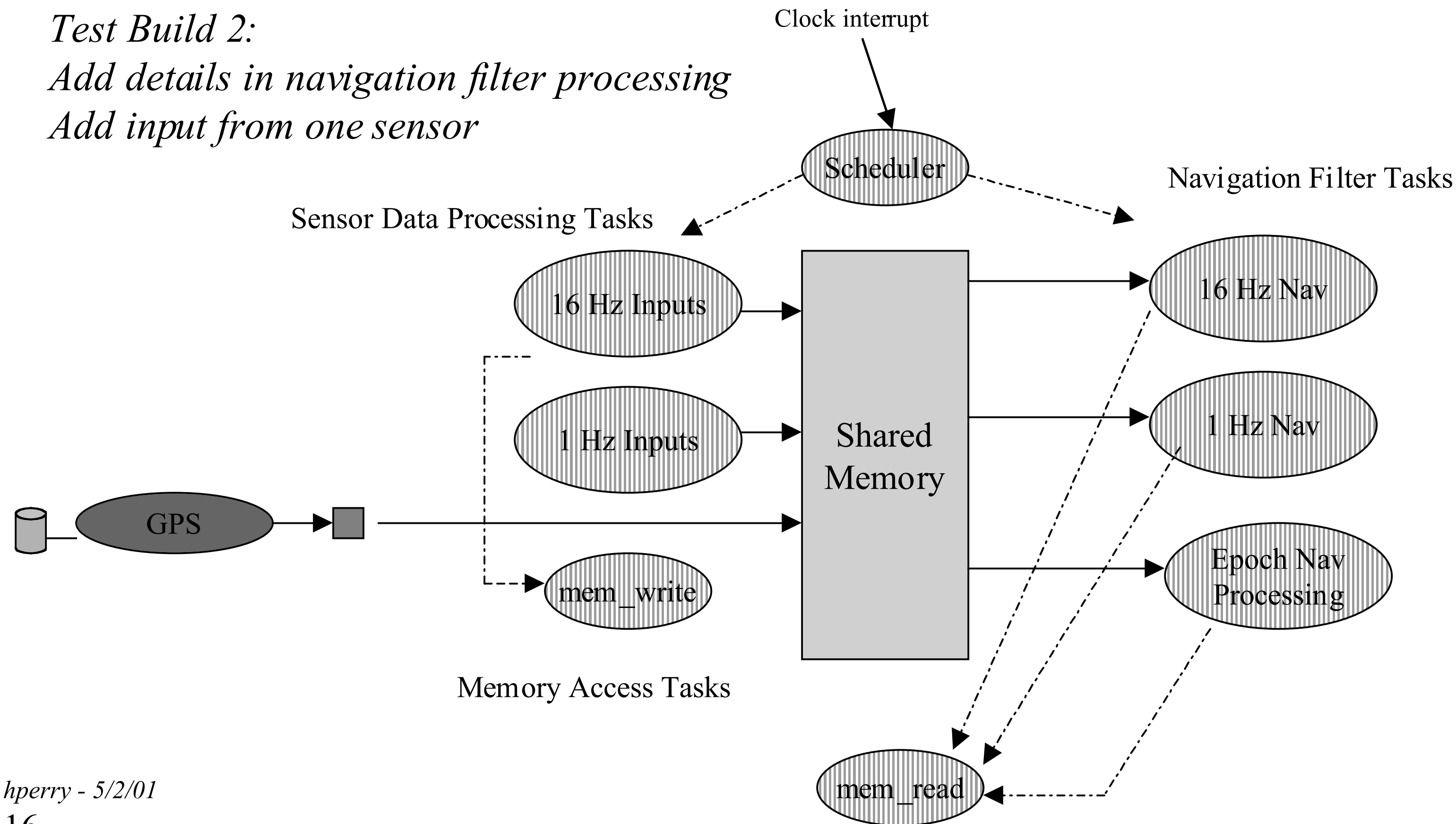


Hardware / Software Integration - An Example

System Unification - The Steps Involved (continued)

3. Determine the best way to fit them together into larger pieces

Test Build 2:
Add details in navigation filter processing
Add input from one sensor

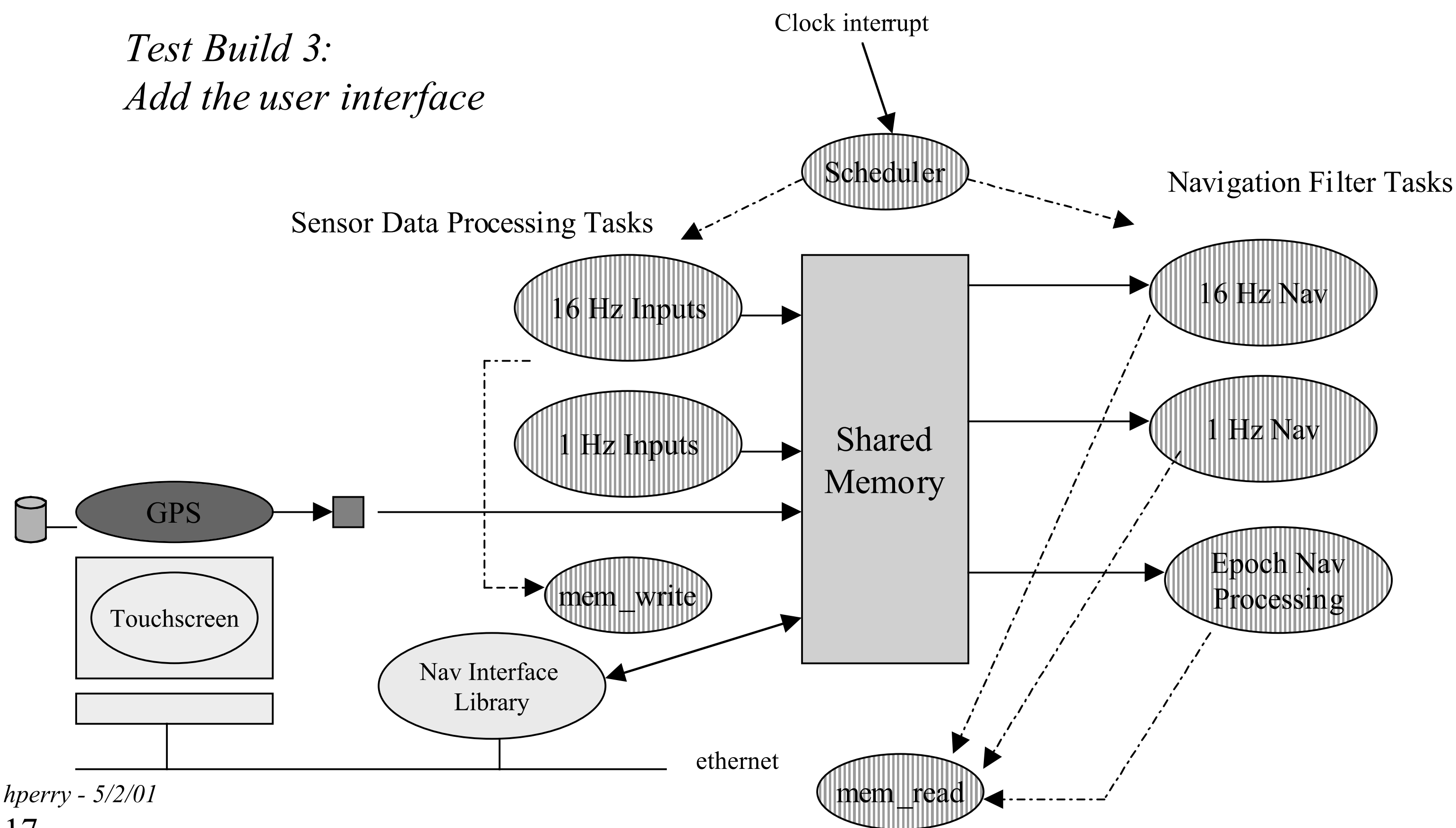


Hardware / Software Integration - An Example

System Unification - The Steps Involved (continued)

3. Determine the best way to fit them together into larger pieces

Test Build 3:
Add the user interface

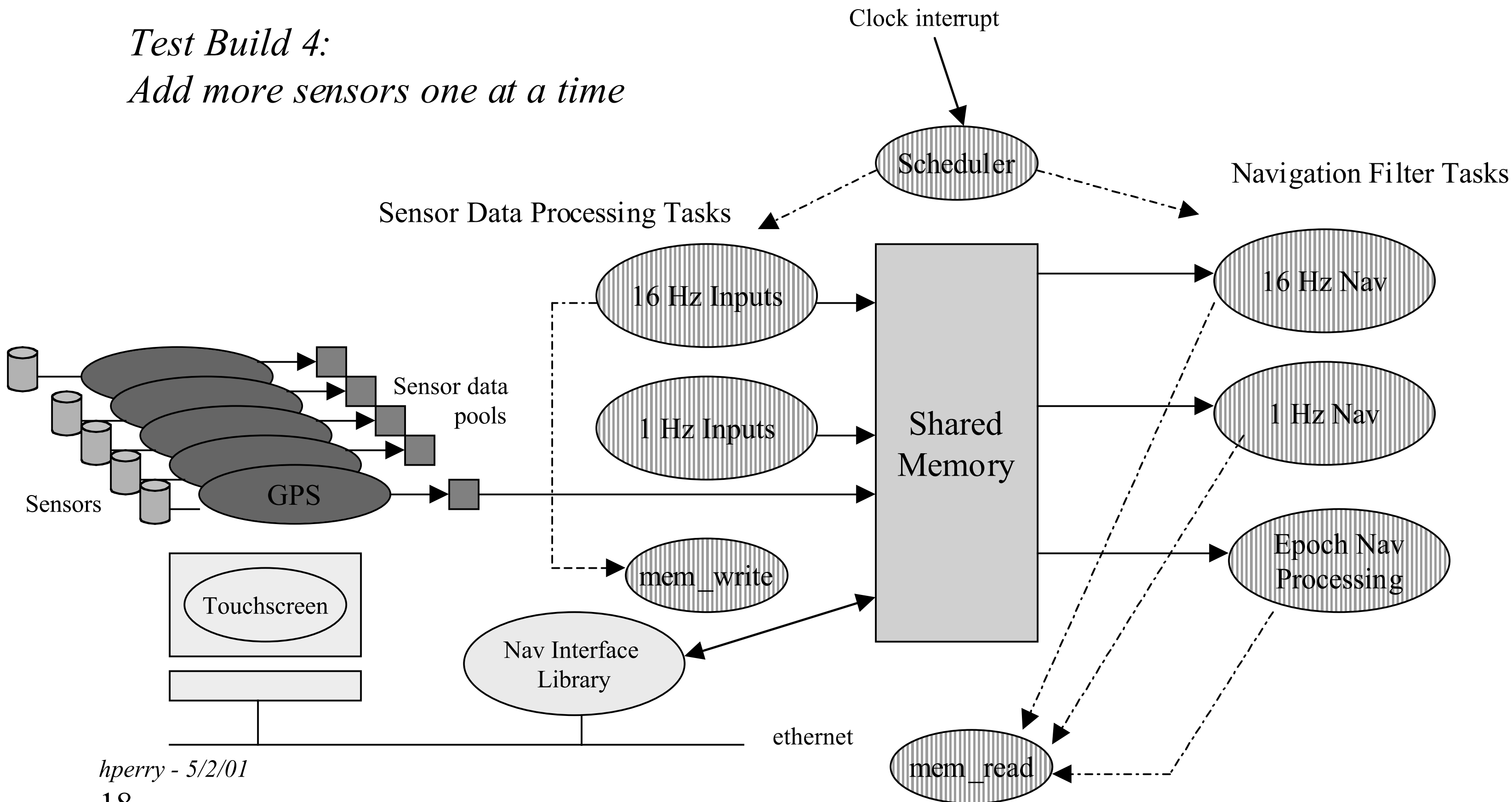


Hardware / Software Integration - An Example

System Unification - The Steps Involved (continued)

3. Determine the best way to fit them together into larger pieces

Test Build 4:
Add more sensors one at a time



Hardware / Software Integration - An Example

System Unification - The Steps Involved (continued)

4. System Testing / Regression Testing

   Lab test -

   - Verification of all data I/O using simulation and real sensors
   - Verification of navigation performance used canned sim scenarios
   - Stress testing (running the system overnight, power cycles, etc.)
   - Verification of navigation system user interface

   Dockside test -

   - Verification of all data I/O and basic navigation using real sensors

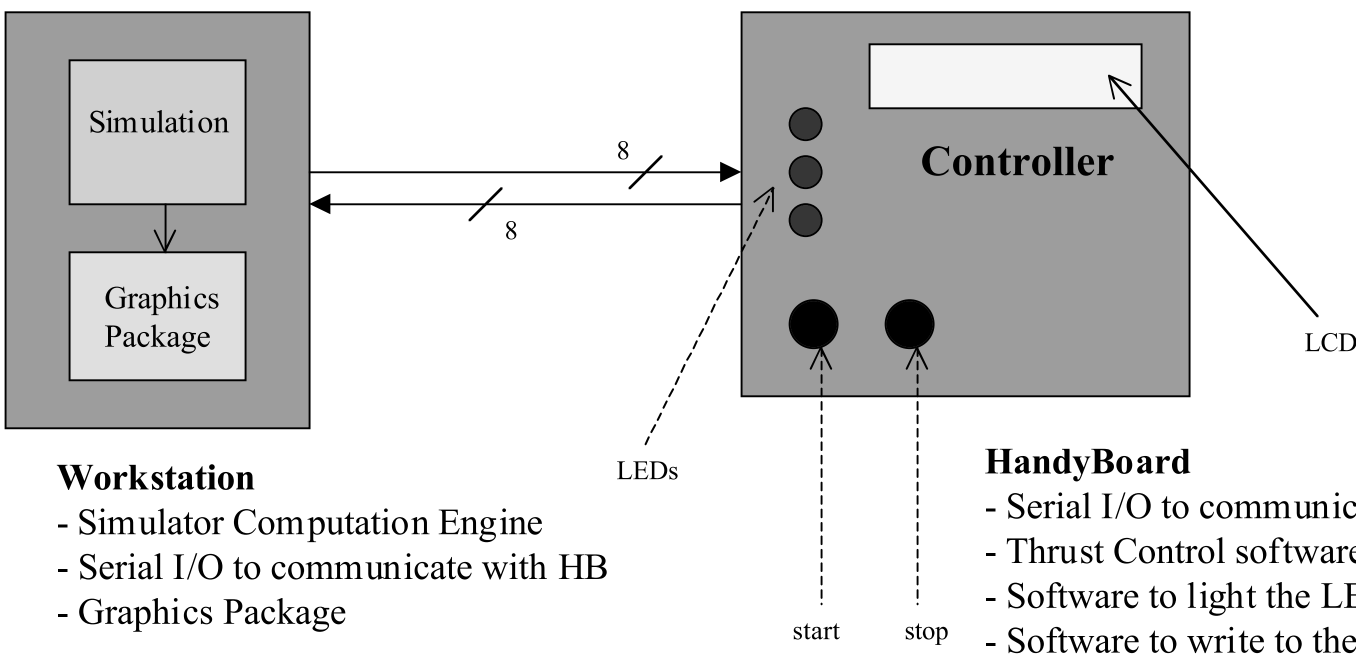
   Sea test -

   - Five days on and below the ocean surface testing nav performance

5. Write up your results - Delivered a test report to the customer

An Example Approach for the MIT16.070 project

1. Establish a baseline - What are the pieces of the system?



Workstation
- Simulator Computation Engine
- Serial I/O to communicate with HB
- Graphics Package

HandyBoard
- Serial I/O to communicate w/ WS
- Thrust Control software
- Software to light the LEDs
- Software to write to the LCD

An Example Approach for the MIT16.070 project

2. What tools will be used for testing?

a. Handyboard (HB)
   - Test code to generate data to fill packet (HB to WS)

b. Simulator (Workstation (WS))
   - Test code to generate data to fill packet (WS to HB)
   - Test Code to print the value of thrust on/off coming from HB
   - Test Code to toggle the “fuel out” bit going to HB
   - Test Code to set altitude and velocity values in data going to HB

An Example Approach for the MIT16.070 project

3. Determine the best way to put the pieces together into a whole system

a. LCD software

- *HB software to write to the LCD*
- Example LCD software test
    - Write a simple standalone program for the HB that will write to the LCD.

An Example Approach for the MIT16.070 project

3. Determine the best way to put the pieces together into a whole system

b. Serial I/O test

- Serial I/O code to communicate with workstation
- Serial I/O code to communicate with Handyboard
- Test code to fill serial packet (WS to HB)
- Test code to fill serial packet (HB to WS)
- *HB software to write to the LCD when data is received*
- Workstation test code using printf() to show data received

- Example Serial I/O test: Test code on WS fills data packet with 11111111, sends it to HB, HB receives data, displays "11111111" on LCD, HB replies with 10001000, WS receives data and prints 10001000 to screen.

An Example Approach for the MIT16.070 project

b. Thrust Controller
   - *Serial I/O software on HB and WS - already tested*
   - On workstation side, modify your previous printf() test code to indicate thrust on/off ()
   - Thrust controller software on HB
   - Example Thrust Controller Test: Using same test code as in the Serial I/O test (but modified to run as a finite loop), the WS fills the data packet with 11111111, sends it to the HB. HB responds by checking thrust controller software and setting the response packet bits accordingly. The WS receives the data packet and displays "thrust on" or "thrust off." Be sure to test all possible combinations of the thrust controller start/stop scenarios.

An Example Approach for the MIT16.070 project

d. Simulator / Graphics Package

- Computation Engine (formula realization for vehicle altitude, velocity and fuel)
  - Try without any HB inputs first. The simulation should run assuming no thrust inputs and should not hang up waiting for the HB data packet. Data is written to telemetry file.
  - Integrate with the graphics package (add function call) Confirm graphics behavior matches telemetry file data.
- *Connect HB: Try with thrust off. Try with thrust on.*
  - *Serial I/O to communicate with Handyboard*
  - *Thrust controller software on HB*

An Example Approach for the MIT16.070 project

e. Test the LEDs
   - WS Test code to toggle the fuel out bit *(or Computation Engine)*
   - *Serial I/O from WS to HB (fuel out bit)*
   - *Serial I/O on HB side (to receive)*
   - Software to light LEDs when fuel is out
   - Example LED Test: Rewrite test code used in the Serial I/O test such that the WS fills the data packet with the fuel out bit set to 1 and sends it to the HB. HB responds by lighting the LED for fuel out. The WS then toggles the fuel out bit to 0 and sends that packet. The HB turns the LED off.
   - You could also use the Computation Engine if you are sure the spacecraft runs out of fuel

An Example Approach for the MIT16.070 project

f. Test the LCD display of Altitude and Velocity
   - *Serial I/O from WS to HB (altitude, velocity)*
   - *Serial I/O on HB side (to receive)*
   - Test code to set altitude and velocity on the Workstation and fill the data packet for the HB (*or Computation Engine)*
   - *Note: You have already tested the ability to display data to the LCD in step "a" in this sequence. Now you are testing to see that data coming in the Workstation serial packet can be accurately displayed on the LCD.*

An Example Approach for the MIT16.070 project

g. Put it all together
   - Remove all test code
   - Compile and link the final load module on the HB and the WS
   - Test the system against the system requirements

An Example Approach for the MIT16.070 project

4. Regression test as necessary (Re-test areas that were affected by fixes)

5. Write up your approach and results in the test report (Problem Set 11)

6. Demonstrate the project (Lab session 5/14-15)

7. Celebrate!!!!

Hardware/Software Integration - Summary

- Hardware/Software Integration is...
  - Uniting modules from multiple sources to form the overall system
  - Verifying the system works
    - according to response time constraints (overall timing)
    - according to the system requirements

- Integrating a real time system is a thought-out process

- The right tools will be essential to the success of your software test effort

- Readings for this lecture: R 13