Putting it all together

MIT 16.070 Lecture 32

Nature of the systems:

- Embedded and real time computing systems have a different focus from "normal" scientific and general-purpose computing.
  - Understanding of electronics, operating systems, and how software works is necessary at a much lower level.
  - A "right" answer is one that is correct and gets there on time. If you run over time, even a right answer is wrong.
  - Because real-time systems are more deterministic than ordinary computing systems, performance can, in many cases, be proven.
  - In cases where it can't be proven, real-time conformance can be enforced with timing interrupts.
Process:

- Timing concerns and reliability are paramount in real-time systems.
  These are addressed through
  - Design
    - Selection of appropriate hardware, operating systems, etc.
      - processing power
      - memory
      - development cost
      - unit cost
      - expected lifetime
    - Construction of software whose operation can be quantified and tested
  - Testing & Debugging
    - Requirements-based
    - Verification vs. validation
  - Optimization

Tools:

- Powerful techniques are available to ensure a system meets its time budget
  - Debuggers
  - Emulators
  - Logic and bus analyzers
  - Simulations and test beds

- The act of testing a system can affect its performance
Operating systems:

- A real-time operating system contains tools and processes that aid in more rapid, easier development of software
  - Tasks and scheduling
  - Interrupt service
  - Inter-task communication

- There are many different sizes and prices for RTOS. Understanding their components will help you determine what, if any, OS features you need

Integration:

- Systems integration is the most painless if your system is composed of many tested subsystems.
- Each component of a system should have its own tests, which should be repeatable by anyone else who comes along.
- Changes to a subsystem require re-testing all other subsystems that might possibly be affected by the change.
What's Important:

- Understand the options for putting together a real time, embedded system
- Select appropriate hardware, software, and system design
- Set up an implementation and test plan
- Understand software engineering enough to assist in development
- Integrate developed and tested components into a final system
WASP Datalogger/SAS

Design, Integration and Test

Scott Rasmussen
March 7th, 2001
Acronyms

WASP – Wide Area Surveillance Projectile
SAS – Stability Augmentation System
GPS – Global Positioning System
SPI – Serial Peripheral Interface
CTM – Counter Timer Module
A/D – Analog to Digital
PIC – A brand of micro-controller
UART – Universal Asynchronous Receiver Transmitter
ISR – Interrupt Service Routine
MCU – Micro-Controller Unit
CPU – Central Processing Unit
RF – Radio Frequency
ms – milli-seconds
UAV – Unmanned Aerial Vehicle
Wide Area Surveillance Projectile

WASP: Quick Reaction, Affordable, “Eyes on Target”

Compatible with Army Artillery or Navy Gun Launch

---

Scott Rasmussen
May 7th, 2001
History

- Master's Student Project at MIT
  - Professor John Deyst
  - Developed concept and performed some initial proof testing

- Aerodynamics Validation at Draper
  - Designed refined to improve aerodynamics
  - Folding and shell packaging design simplified

- First Flight Crashed
  - Flight tests began January 2000
  - Data logging system requested for use during further integration and test phase flights
  - Parachute recovery mechanism

Scott Rasmussen
May 7th, 2001
Requirements

- **Sense and Log ‘Aircraft State’**
  - at a minimum it is desirable to be able to perform post-flight analysis to determine what ‘went wrong’ in the event of an unsuccessful flight test

- **Stability Augmentation System (SAS)**
  - slow down the aircraft’s flight dynamics to permit flight by a remote human pilot

- **Telemetry Downlink**
  - in the event of onboard data log failure, provide a record of at least part of the onboard state information
  - enable in-flight functionality checks

Assume existence of flight vehicle with given actuators and servo mechanisms

May add sensors, electronics and batteries to suit
Sensors and Outputs

- **Pilot Commands:** ~70Hz
- **Analog Measurements:** ≥ 1 per cycle
  - Roll, Pitch and Yaw Rate gyros
  - Engine RPM
  - Airspeed
  - System Battery Voltages
- **Global Positioning System Measurements:** 1Hz
- **Debug/Testing Control:** Sporadic

- **Data Log to Non-Volatile Storage:** Full Log
- **Stability Augmentation Commands:** ~70Hz
- **Telemetry Data:** >1Hz
- **Debug/Testing Output:** Sporadic

Scott Rasmussen
May 7th, 2001
Stability Augmentation

- PID - Proportional, Integral, Derivative Control
- Three Loops
  - Aileron $\Leftarrow$ Roll and Yaw Rate
    $$ da = \frac{-0.05(s+6)}{s} (p_{err} + r_{err}) $$
  - Elevator $\Leftarrow$ Pitch Rate
    $$ de = \frac{-0.01(s+23)}{s} q_{err} $$
  - Rudder $\Leftarrow$ Yaw Rate
    $$ dr = -0.08 r_{err} $$
  - No Throttle Feedback
Design Considerations

- Design choices - not necessarily ‘correct’
- Overriding principle - do it simply and quickly
  - First way something worked was generally used, little refinement
  - Heavy reliance on past experience and intuition

- Development Language: C and Assembly
- Code Optimizations: Assembler Level Device Drivers
- Operating System: PicoDOS - microkernel
- Prioritization/Scheduling: In Chip Interrupt Handling
- Intertask Communications:
  Global Memory, Priority Ceiling Protocol
- Testing: Development and Test Interwoven
Software Design

- Three ‘independent’ processors
  - MCU: Micro-Controller Unit
  - CTM: Counter Timer Module
  - SPI: Serial Peripheral Interface
  - Complete avionics uses at least 8 MCU’s

- Complex Interrupt Handling
  - Multi-Level prioritization of Interrupts
  - Interrupts can interrupt lower priority interrupts
  - Elementary Interrupt queueing
  - Tasks are both prioritized and driven by interrupts

- Main loop runs continuously
  - Runs tasks to Poll certain events
Main Loop

oldSW = !SafetySwitchIsAuto;
oldDL = !DataLoggingIsOn;
while (!userCancel) {
    //SetPFP7();
    WaitForInterrupt();
    //ClearPFP7();

    if (oldSW != SafetySwitchIsAuto) {
        if (SafetySwitchIsAuto) {
            printf("Auto\n");
        } else {
            printf("Manual\n");
        }
    oldSW = SafetySwitchIsAuto;
}

if (oldDL != DataLoggingIsOn) {
    if (DataLoggingIsOn) {
        printf("Data Logging On\n");
    } else {
        printf("Data Logging Off\n");
    }
    oldDL = DataLoggingIsOn;
}

if (DataReadyForWrite) {
    if (DataLoggingIsOn) {
        // Write the data in the buffer to the log file
        write(fd_logFile, (void *)chInBak, 16*BufSize*2*2);
    }
    // Done writing so reset the write flag
    DataReadyForWrite = false;
}

ParseGPS();

if (GPSDataReadyForWrite) {
    if (DataLoggingIsOn && dataFileExists) {
        // Write the data in the GPS message buffer to the log file
        write(fd_logFile, (void *)gpsDataBuf, 32*2);
    }
    // Show the number of satellites in view
    printf("%d", ((gpsInternal*)gpsDataBuf)->svs);
    // Done writing so reset the write flag
    GPSDataReadyForWrite = false;
}

fflush(NULL);
Telemetry Ground Station
Testing and Integration

- Design, Development and Implementation Testing
  - Software built up piece-wise
  - Complete prototype system

- Unit Level Testing
  - Bench tests for GPS
  - Rate table tests for Gyros

- Flight Testing
  - Glide Test
  - Straight ahead flight test
  - Turning flight

- System Bench Testing
  - Confirm that all systems work together

- Bench tests for Airspeed
  - Wind Tunnel tests for Airspeed
  - Bench tests for Pilot command input and servo output

- Bench tests for RF Modem system

- Parachute recovery test
Pre-Flight Check
Crash Recovery

Scott Rasmussen
May 7th, 2001