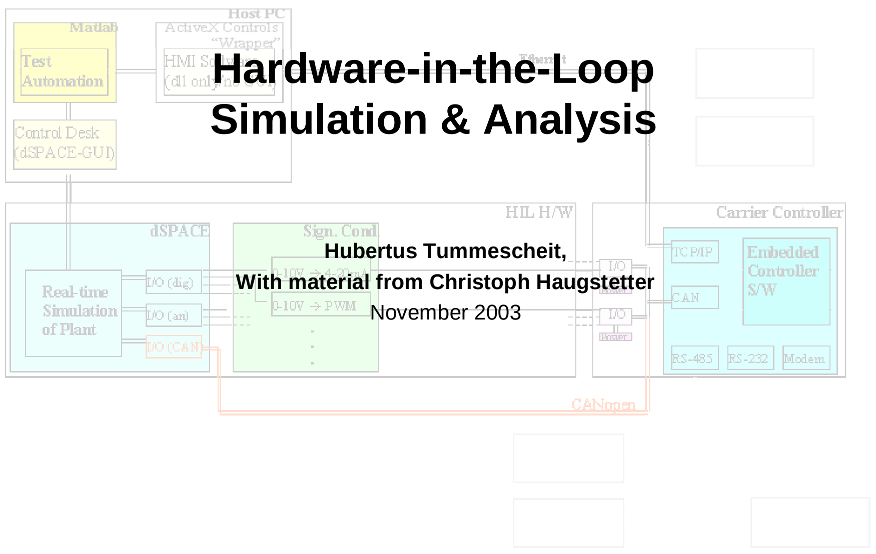


Hardware-in-the-Loop Simulation & Analysis

Hubertus Tummescheit,
With material from Christoph Haugstetter
November 2003

Outline

- Achieving Real-time performance
  - Numerical and symbolic techniques to achieve real-time performance
    - Fixed step solvers: implicit or explicit methods
    - Tearing, inline integration
    - Mixed-mode integration
    - Mode handling
- Hardware-in-the-loop definitions
  - Scope – what it is, what it is not
  - Motivation
- Illustrate HIL effectiveness by example
  - Choosing the right model for the task
  - I/O scaling verification
  - Start / stop verification
  - Dynamic performance validation

Fixed Step Solvers:

Simple 1st order ODE example:

$$\dot{x} = f(x)$$

Discretize using explicit Euler:

$$x_{n+1} = h \cdot f(x_n)$$

Discretize using implicit Euler:

$$x_{n+1} = x_n + h \cdot f(x_{n+1})$$

At each time step, solve this equation for x_{n+1}

Fixed Step Solvers:

At each time step, solve this equation for x_{n+1}

Similar for systems of equation, but for implicit methods, use local linearization (Jacobian)for some steps instead of non-linear function:

$$A = \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \cdots & \frac{\partial f_1}{\partial x_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial f_1}{\partial x_1} & \cdots & \frac{\partial f_n}{\partial x_n} \end{bmatrix} \approx f(x_{n+1})$$

Computation time:

1. Step update (constant)
2. Update of Jacobian if needed (much longer than 1)

Fixed Step Solvers:

Why use implicit solvers?

- Explicit methods are unstable for stiff systems, except for very small step sizes.
- Implicit solvers are stable for much larger step sizes for stiff systems
- Therefore, implicit solvers may be much faster in spite of the equation system
  and the time to update the Jacobian

Tearing

Tearing of linear or non-linear system:

$$g(x) = 0$$

1. Split the vector of unknowns into two parts called tearing variables, x_t, and the remaining variables, x_r, and
2. Select some equations of g as as residue equations such that the residues can be determined if the tearing variables are known

$$g_1(x_r, x_t) = 0$$
$$g_2(x_r, x_t) = residue(x_t)$$

$\rightarrow$ Usually sparse system of dimension dim(g_1)+ dim(g_2) is reduced to a (usually full) system of of dimension dim(g2).

Tearing Example

Incidence matrix

$$\begin{array}{cc|cccccc|c|c}
 & & a & b & c & d & e & f & & \\
g_1 & h_1 & 1 & & & & & * & & 0 \\
g_1 & h_2 & * & 1 & & & & * & & 0 \\
g_1 & h_3 & & * & 1 & & & * & = & 0 \\
g_1 & h_4 & * & * & * & 1 & & * & & 0 \\
g_1 & h_5 & & * & * & & 1 & * & & 0 \\
g_2 & h_6 & & * & & * & * & * & & 0
\end{array}$$

Simple to solve for a b c d and e when f is known. Assume f is known. Then we can present the problem h_6 (f) = 0 to a numerical solver. Computationally hard to find good tearings. System for a,b,c,d,e might even be linear

- Combinatoric explosion: NP-hard problem
- Pivoting to find structure?
- Users may have insight (“false dynamics”), automatic in Dymola

Tearing for linear systems

$$\begin{pmatrix} \mathbf{L} & \mathbf{A}_{12} \\ \mathbf{A}_{21} & \mathbf{A}_{22} \end{pmatrix}\begin{pmatrix} \mathbf{x}_r \\ \mathbf{x}_t \end{pmatrix} = \begin{pmatrix} \mathbf{b}_1 \\ \mathbf{b}_2 + \mathbf{residue}(\mathbf{x}_t) \end{pmatrix}$$

In the linear case above, where L is required to be a lower-triangular, non-singular matrix, the result is even nicer. The system can be transformed to

$$(\mathbf{A}_{22} - \mathbf{A}_{21}\mathbf{L}^{-1}\mathbf{A}_{12})\mathbf{x}_t = \mathbf{b}_2 - \mathbf{A}_{21}\mathbf{L}^{-1}\mathbf{b}_1$$

L is lower triangular, so inversion of L is just backward Substitution and can be done symbolically.

Inline Integration

Back to implicit solvers:

Can tearing be combined in a smart way with implicit solvers?
Linear case:

$$\mathbf{x}_{\mathbf{n+1}} = \mathbf{x}_{\mathbf{n}} + h \cdot \mathbf{A} \cdot \mathbf{x}_{\mathbf{n+1}}$$

A often large and sparse, same dimension as number of states.
Procedure: perform the discretization symbolically before handing the system to the solver and perform tearing to get smaller system (the discretization formula is in-lined into the equations)

Mixed-Mode Integration

Idea to combine advantages of implicit and explicit solvers.
Split system into fast and slow part:

$$\begin{aligned} \dot{\mathbf{x}}^S &= \mathbf{f}^{\mathbf{S}}(\mathbf{x}^{\mathbf{S}}, \mathbf{x}^{\mathbf{F}}) & \qquad \mathbf{x}_{\mathbf{n+1}}{}^{\mathbf{S}} &= \mathbf{x}_{\mathbf{n}}{}^{\mathbf{S}} + \mathbf{h} \cdot \mathbf{f}^{\mathbf{S}}(\mathbf{x}_{\mathbf{n}}{}^{\mathbf{S}}, \mathbf{x}_{\mathbf{n}}{}^{\mathbf{F}}) \\ \dot{\mathbf{x}}^F &= \mathbf{f}^{\mathbf{F}}(\mathbf{x}^{\mathbf{S}}, \mathbf{x}^{\mathbf{F}}) & \qquad \mathbf{x}_{\mathbf{n+1}}{}^{\mathbf{F}} &= \mathbf{x}_{\mathbf{n}}{}^{\mathbf{F}} + \mathbf{h} \cdot \mathbf{f}^{\mathbf{F}}(\mathbf{x}_{\mathbf{n+1}}{}^{\mathbf{S}}, \mathbf{x}_{\mathbf{n+1}}{}^{\mathbf{F}}) \end{aligned}$$

Analyze linear situation. Use $\mathbf{P} = diag(\delta_1, ..., \delta_n),\ \delta \in \{0, 1\}$
To split up system into two parts, and discretize the fast part with the implicit Euler, the slow part with the explicit Euler method

$$\begin{aligned} \mathbf{x}_n^S &= \mathbf{P}\mathbf{x}_n^S + \mathbf{hPA}\mathbf{x}_n^S \\ \mathbf{x}_{n+1}^F &= (\mathbf{I} - \mathbf{P})\mathbf{x}_n^S + \mathbf{h}(\mathbf{I} - \mathbf{P})\mathbf{A}\mathbf{x}_{n+1}^S \end{aligned}$$

Mixed-Mode Integration

$$\mathbf{x}_n^S = \mathbf{P}\mathbf{x}_n^S + \mathbf{h}\mathbf{P}\mathbf{A}\mathbf{x}_n^S$$
$$\mathbf{x}_{n+1}^F = (\mathbf{I}-\mathbf{P})\mathbf{x}_n^F + \mathbf{h}(\mathbf{I}-\mathbf{P})\mathbf{A}\mathbf{x}_{n+1}^F$$

Add the equations and solve for x_{n+1}. For given step size h

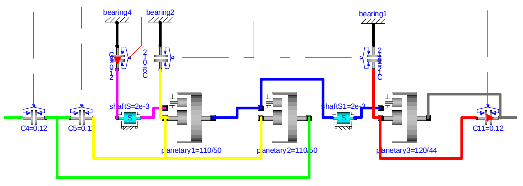
$$\mathbf{x}_{n+1} = \mathbf{U}_h\mathbf{x}_n$$
$$\mathbf{U}_h = (\mathbf{I}-\mathbf{h}(\mathbf{I}-\mathbf{P})\mathbf{A})^{-1}(\mathbf{I}+h\mathbf{P}\mathbf{A})$$

Problem (tough!): split into fast and slow part to make h as large as possible such that U_h is stable.

Details see Schiela, Olsson: Mixed-mode integration for Real-time simulation, Modelica Workshop 2000

Mode handling

- Automatic gearbox example
- Efficient solution of linear systems of equations
- Coefficients depending on clutch switching
- Different code for each combination
- All different combinations: 2^n
- Instead, determine used combinations of switches (modes) by off-line simulation
- Speed-up: 10 times



Mode handling II

- Set Advanced.ModeHandling = true
- Translate
- Simulate typical cases
- Translate generated model with mode information (see Dymola Manual for details)
- Simulate

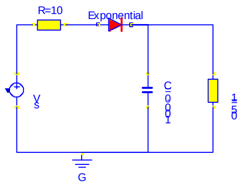
Generated Model with Modes

```
model geartestModeValues

  extends geartest;

  Boolean bigModeVariables[:] = {B4.Locked, B3.logic.Locked, B1.logic.Locked, B2.logic.Locked,
     C2.logic.Locked, C1.logic.Locked, C0.Locked, B0.logic.Locked, freeWheel.Locked,
     LU.logic.Locked, B5.logic.Locked};

  Real bigModes[13,11]={
    { 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
    { 1, 0, 0, 0, 0, 0, 1, 0, 1, 0, 1},
    { 1, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0},
    { 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0},
    { 1, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0},
    { 1, 0, 0, 0, 1, 0, 1, 1, 0, 0, 0},
    { 1, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0},
    { 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0},
    { 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0},
    { 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1},
    { 1, 0, 0, 0, 1, 0, 1, 0, 1, 0, 0},
    { 1, 0, 0, 0, 1, 0, 1, 1, 1, 0, 0},
    { 1, 0, 0, 0, 1, 0, 0, 1, 1, 0, 0} };

end geartestModeValues;
```

Code Structure

```
if (B4_Locked == 0 AND B3_logic_Locked == 0 AND B1_logic_Locked == 0 AND
    B2_logic_Locked == 0 AND C2_logic_Locked == 0 AND C1_logic_Locked == 0
    AND C0_Locked == 0 AND B0_logic_Locked == 0 AND freeWheel_Locked == 0
    AND LU_logic_Locked == 0 AND B5_logic_Locked == 0) {

    // Solve simpler system of equations for this mode
} else {
```

Ideal diode with Modes



```
OutputSection
DynamicsSection
Vs_u_v = A_0*sin(6.28318530717959*f_0*Time)+v0_0;
Vs_p_v = Vs_u_v;
D_n_v = C_v;

MixedSystemOfEquations(5){
  /* Mode handling of an equation system */
  if (D_off == 0) {
    D_v = 0;
    R_n_v = D_v+D_n_v;
    R_v = Vs_p_v-R_n_v;
    Vs_n_i = 0.1*R_v;
    D_s = Vs_n_i;
  }
  else {
    Vs_n_i = 0;
    R_v = 10*Vs_n_i;
    R_n_v = Vs_p_v-R_v;
    D_v = R_n_v-D_n_v;
    D_s = D_v;
  }

  MixedModeStartBoolean //
    UpdateVariable(D_off, Less(D_s, 0, 0));
  ...
```

```
InitialSection
PI_0 = 3.14159265358979;
Vs_n_v = 0;
G_p_v = 0;
C_n_v = 0;
R1_n_v = 0;
```

Speed-up with Modes

- Simulation of gearbox with 11 clutches on dSPACE (500 Mz Dec alpha processor)

- Linear algebraic loop with 55 unknowns,
  Tearing gives 23 unknowns
- Cycle time: > 2 ms

- Mode handling: 10-30 active modes
- Linear systems of sizes: 0-9
- Constant J matrices; precalculation and LU-factorization
- Maximum cycle time: 0.18 ms

Speed-up with Modes

- Simulation of gearbox on dSPACE

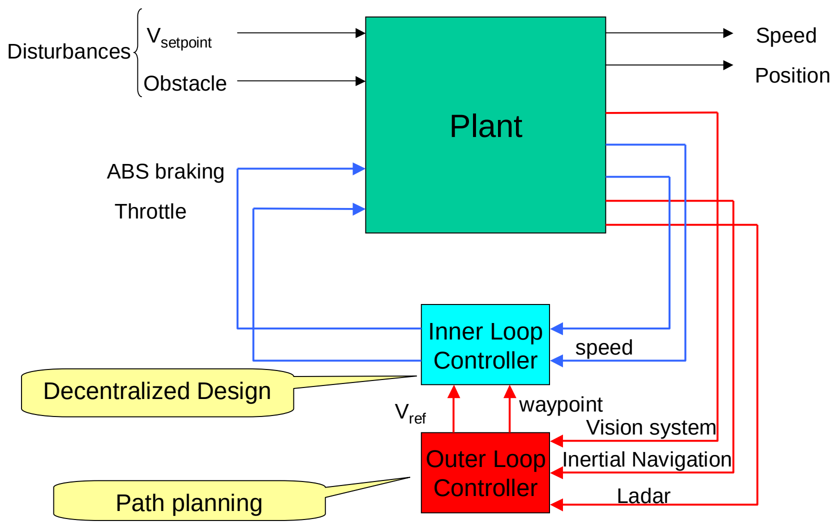
- Previous average cycle time: 0.9 ms
- Previous maximum cycle time: 2.4 ms
- Previous initial set-up time: 2.5 ms

- Average cycle time: 0.15 ms
- Maximum cycle time: 0.18 ms
- Initial set-up time: 0.4 ms

Part II: Industrial Examples of HIL-testing

Most critical Control Loops

Complex controls often have cascade architecture



Ensure a common vocabulary

Definitions

- **Real-time model**: computing a time interval Δt in simulation < Δt; predictable performance
- **Validation**: “Did we build the right system?” (is the problem solved correctly)
- **Verification**: “Did we build the system right?” (does solution conform to specs)

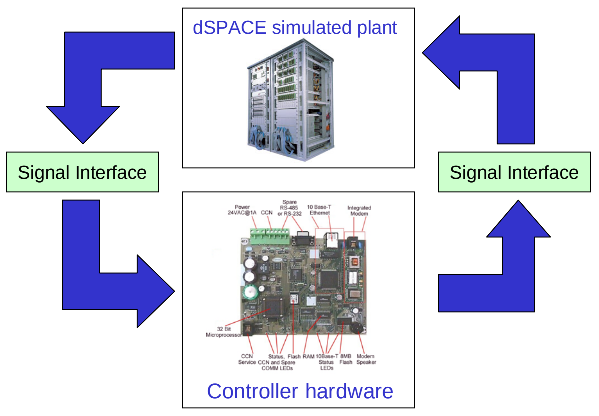
- **dSPACE**: manufacturer of real-time simulator
- **ECU**: Electronic control unit (“plant controller”)
- **HIL**: Hardware-in-the-Loop: Plant simulated in real-time, connected to real ECU
- **H/W**: Hardware
- **Regression testing**: assure correct performance after a S/W change
- **Signal conditioning**: conversions between different signal types; scaling/filtering
- **Simulator**: Rack-mounted setup, containing dSPACE computer, signal conditioning, break out box; connected to Host PC (non-realtime)
- **S/W**: Software

HIL definitions

A powerful, flexible tool for verification & validation of controller performance

Definition. Process and tools for verifying the ***logical*** and ***temporal*** correctness of integrated control system hardware & software.

- *Control loops closed using real-time simulated plant.*
- *Interfaces are exercised to ensure correct system integration.*
- *Testing can be automated (Design for Experiments).*

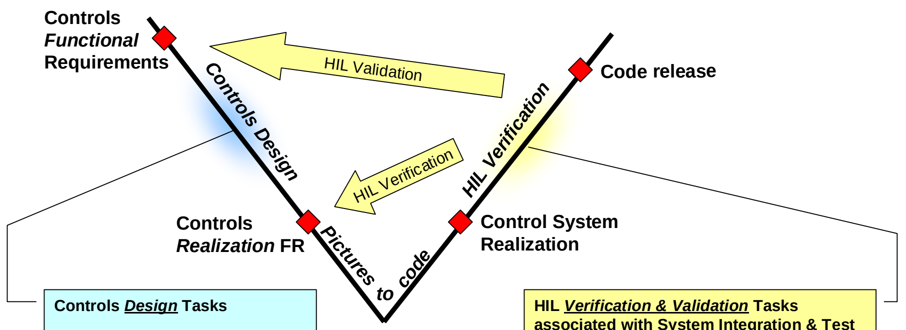


HIL can also be used for:

- Automated production test
- Field service debug & test
- Calibration

Control System Development Process

HIL key to Validation & Verification of controlled dynamic performance



Controls *Design* Tasks

- Process control architecture
- Compensator design e.g. lead/lag, PID, selectors
- Start / stop design
- Alarms, shutdowns, diagnostics
- Communications software

HIL *Verification & Validation* Tasks associated with System Integration & Test

- Closed-loop, real-time, dynamic performance
- Start / stop verification
- Signal conditioning scaling
- Connector pin-out
- Calibration of gains

HIL is a flexible, rigorous validation & verification tool that complements "desktop" design methods, process & tools.

Controls implementation / Test plan Risks

Controller software / functionality being validated

- Morning stories (example from UTCFC early prototype of fuel cell) cite "software" bugs almost *daily (some of them error in the Functional Requirements, not S/W !)*
- Majority of "bugs" relate to VALIDATION not VERIFICATION
- HIL testing would catch many safely

June 23:

*Testing resumed today with a light-off that triggered a high temperature shutdown on the …. The **cause of the shutdown was an unexpected transition into auto fuel/air control because of a missing time delay override**. Subsequent attempts to restart revealed … onto the high shift catalyst. Based on this, a decision has been made to replace the … catalyst and preparations for this replacement were made during second shift.*

June 12:

*Yesterday's test showed that software **skipped few states during normal shutdown**. Correction has been made to the software, and the new version has been checked out during second shift. It will be loaded to the test stand computer tomorrow morning.*

June 11:

*Fuel and air calculation problem seen last night was **due to equation problem not software. Software was re-written with the correct calculation**. Transition from startup leg to main leg was tested successfully.*

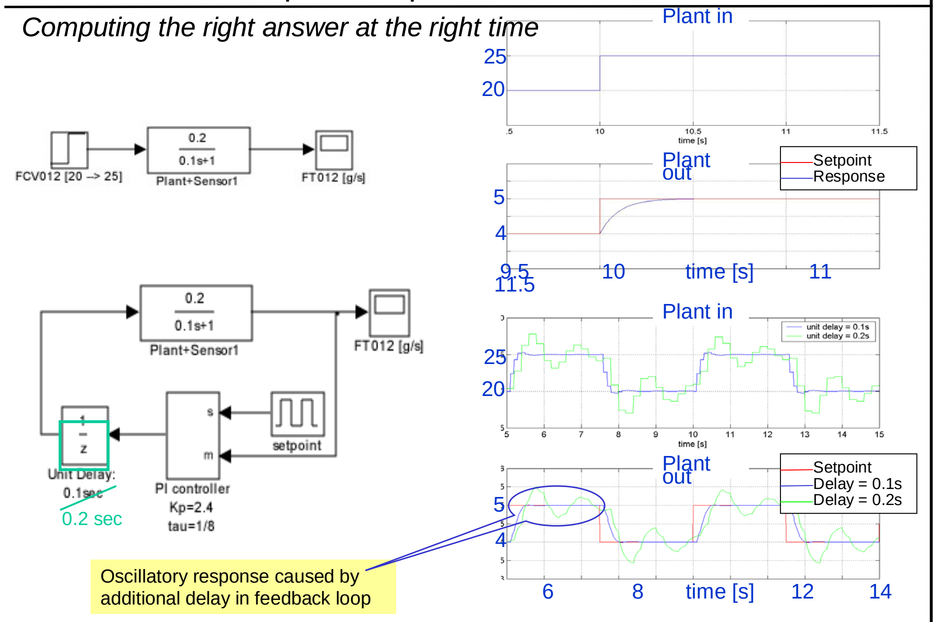
***New Software issue was discovered when performing normal shutdown**. During normal shutdown, the air flow should stop first to prevent …, but current shutdown sequence stops air and fuel concurrently. No large … spike was noticed on last shutdown. The team can manually drive down air flow during shutdown sequence if necessary. Software team are working hard to determine root cause and correct this issue.*

June 10:

*The primary objective of todays' testing was to advance through the transition from the start fuel leg to fuel flow control using the main fuel valve (without overriding the FCV and then to hold in R65 and switch to automatic air flow control. Just like before, we overrode the FCV because **the fuel flow measurement was still reading 10 times greater than the actual flow** as measured from the test stand. This discrepancy is an issue with the software calculation which will be fixed tomorrow. The transition to main fuel valve was smooth.*

Motivational Example: Temporal Correctness

Computing the right answer at the right time



Why HIL? Why Now?

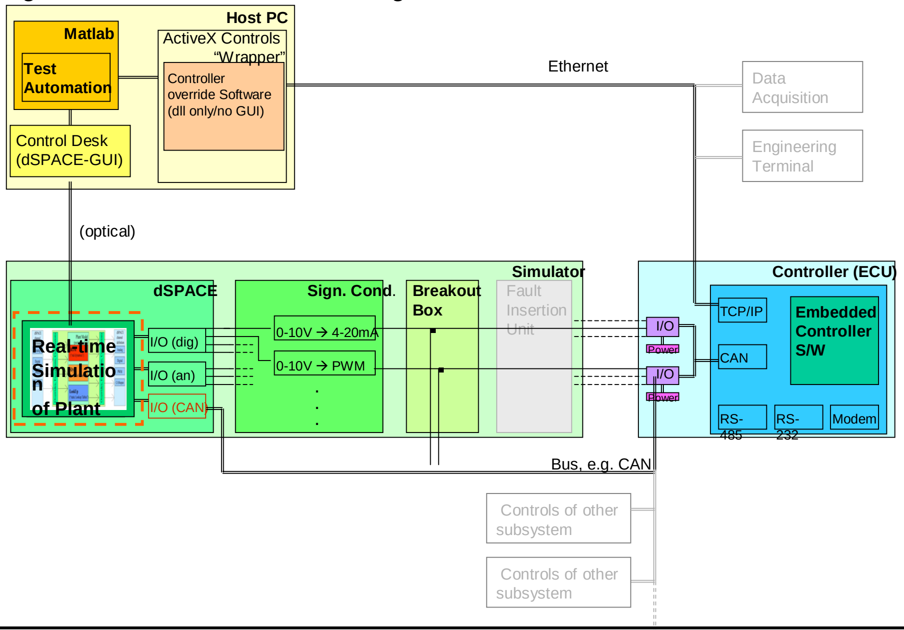
Prototype software & hardware increase risk to schedule and performance

Driver	**Consequence**
Controller hardware is pre-production	H/W performance uncertain
Controller bandwidth approaching sample rate	Logical vs. temporal correctness critical
S/W checkout requires 5 man-weeks	Limited regression testing done
Limits of dynamic analysis (“desktop analysis”) due to various feasibility constraints	- Exception handling not thoroughly tested - Quantization errors, electrical noise, time delays neglected
Evaluation of new control architectures	Plant downtime, during switch-over
Multiple ECUs	(Bus) communication critical to performance
Faster design cycles (“concurrent engineering”)	Reduced integration time, shorter turn-back cycles

It is often the ***Combination*** of several effects that causes poor closed-loop performance e.g. limit cycles (oscillations)

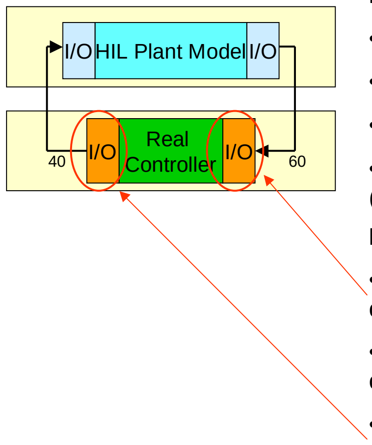
Example High-Level H/W Setup

Signal Flow / Communication Diagram



HIL Example 1: Scaling Tests

Static model sufficient



Model characteristics

- "Low-level Verification (against FR)"
- Static models: potentially very fast
- Override time delays in sequence (→ speed up)
- All test results can be saved, together with report (traceable, documented verification)

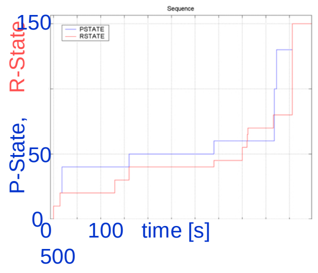
Model applications

- Override plant model Out → Read internal controller variables (through Ethernet)
- Same setup: compare controller-internal, calculated variables against pre-established table
- Override internal controller variables → Read plant model Inputs
- Compare controller parameters against FR

Morning Story 6/9/2003: "... measurement was reading about 10 times greater than the actual flow as measured from the test stand. This discrepancy is a software issue ..."

HIL Example 2: Sequence (Start/Stop)

Simplified Dynamics enable robust and fast performance



Model characteristics

- Mostly based on first-order dynamics, covering main cross-correlations
- Suitable for start-up: radically changing physics
- Tailored to specific test objective (wide range)
- ***Not*** a detailed dynamic analysis

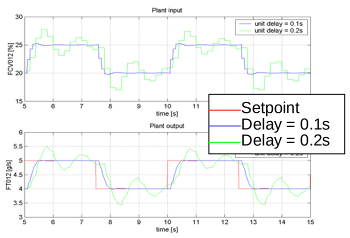
Model applications

- Verify Truth-tables, sequential correctness
- Start-value settings
- Alarms & Shutdowns in each state

Morning Story 5/22/2003: *"… the associate* felt an electric shock entering the right side of his body … indicated that 277 VAC was present …"
Morning Story 5/27/2003: "Heaters are on when emergency shut down was triggered, it was suspected as a software issue"

HIL Example 3: Dynamic Performance

Linear Model provides high (local) fidelity



Model characteristics

- High level of dynamic fidelity of interest
- Covers second-order effects
- Fidelity is best close to linearization point

Model applications

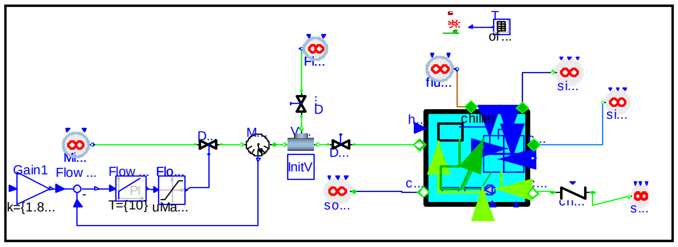
- Verify performance of auto-tuning
- Advanced (e.g. multivariable) control concepts
- Stability of fast inner loops; timing issues
- Interaction with other subsystem controllers

Limits of Performance 5/20/2003: *even small delays or imperfect tuning hurt transient performance considerabl*

Combined Heat and Power HIL Activity

Modeling / Hardware

- Dynamic models created in Dymola by SJTU (China)
  - Based on ThermoFluid Library (Eborn/Tummescheit)
  - Dynamics of system are slow enough for HIL (30-40 states, <1000r/s)
  - Non-linear equation sets are too difficult for HIL (can be fixed)…~15 numerical Jacobians!
  - Linearized single-operating-point models currently used
- dSpace single processor system used for Chiller emulation
  - Dymola -> Simulink interface to dSpace currently used
  - Dymola -> dspace interface under evaluation
- Control system re-constructed for HIL experiments
  - 40 inputs, 2 variable outputs, 8 relay outputs



Industry Example: Test Plans and Automation

Utilize the full potential: how we can employ the tool most efficiently

- Enable precise, repeatable, recorded, standardized work-flow: S/W quality
- Design for Experiment: rigorous, wide range of test scope
- Execution speed much faster without “Human-in-the-Loop”
- Implemented in Matlab (alternative would be Python)
  - flexible to handle exceptions
  - includes the test documentation
- Goals:
  - Standard regression testing in < 2 hour
  - Eliminate S/W and H/W turn-backs