

Auto-generated production code development for Ford's Fuel Cell Vehicle Programme

Charlie Wartnaby, Pi Technology

C.E. Wartnaby, S.M. Bennett* and M. Ellims

Pi Technology, Milton Hall, Ely Road, Milton, Cambridge, England CB4 6WZ

R.R. Raju, M.S Mohammed, B. Patel and S.C. Jones

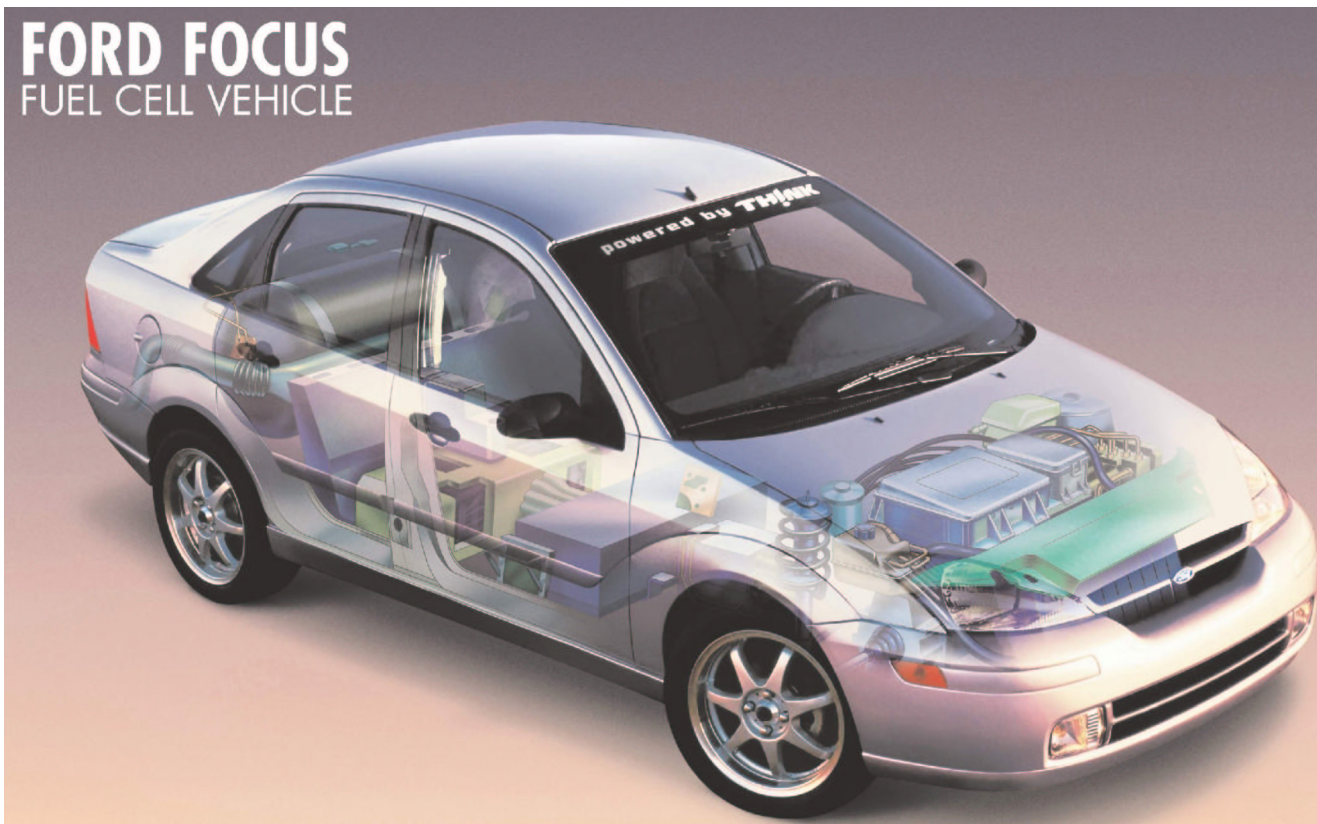
Ford Motor Company, 15000 Commerce Drive North, Dearborn, Michigan 48120-1261, USA

**S.M. Bennett now at Accurate Technologies, Inc*



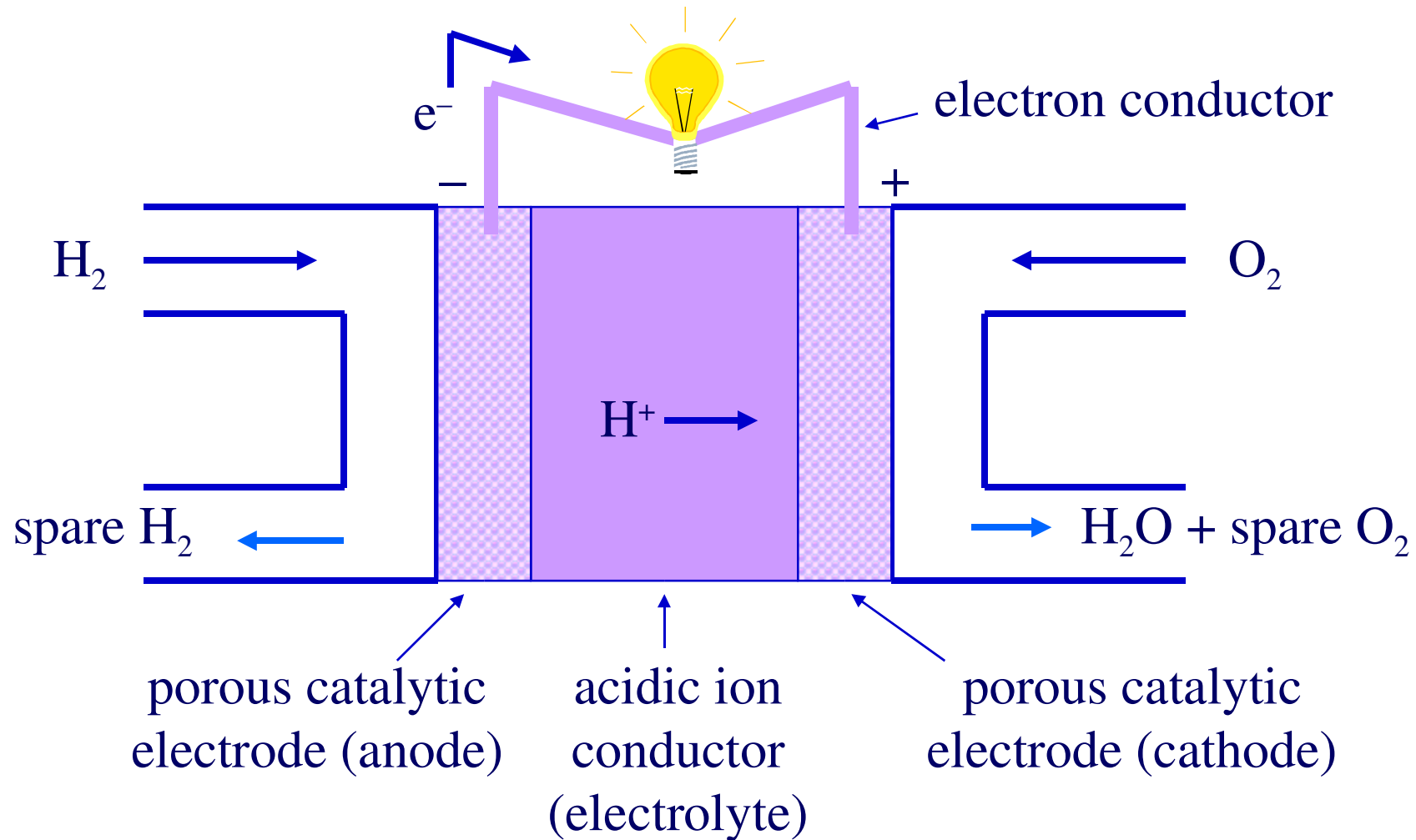
<http://www.pitechnology.com>

FORD FOCUS FUEL CELL VEHICLE

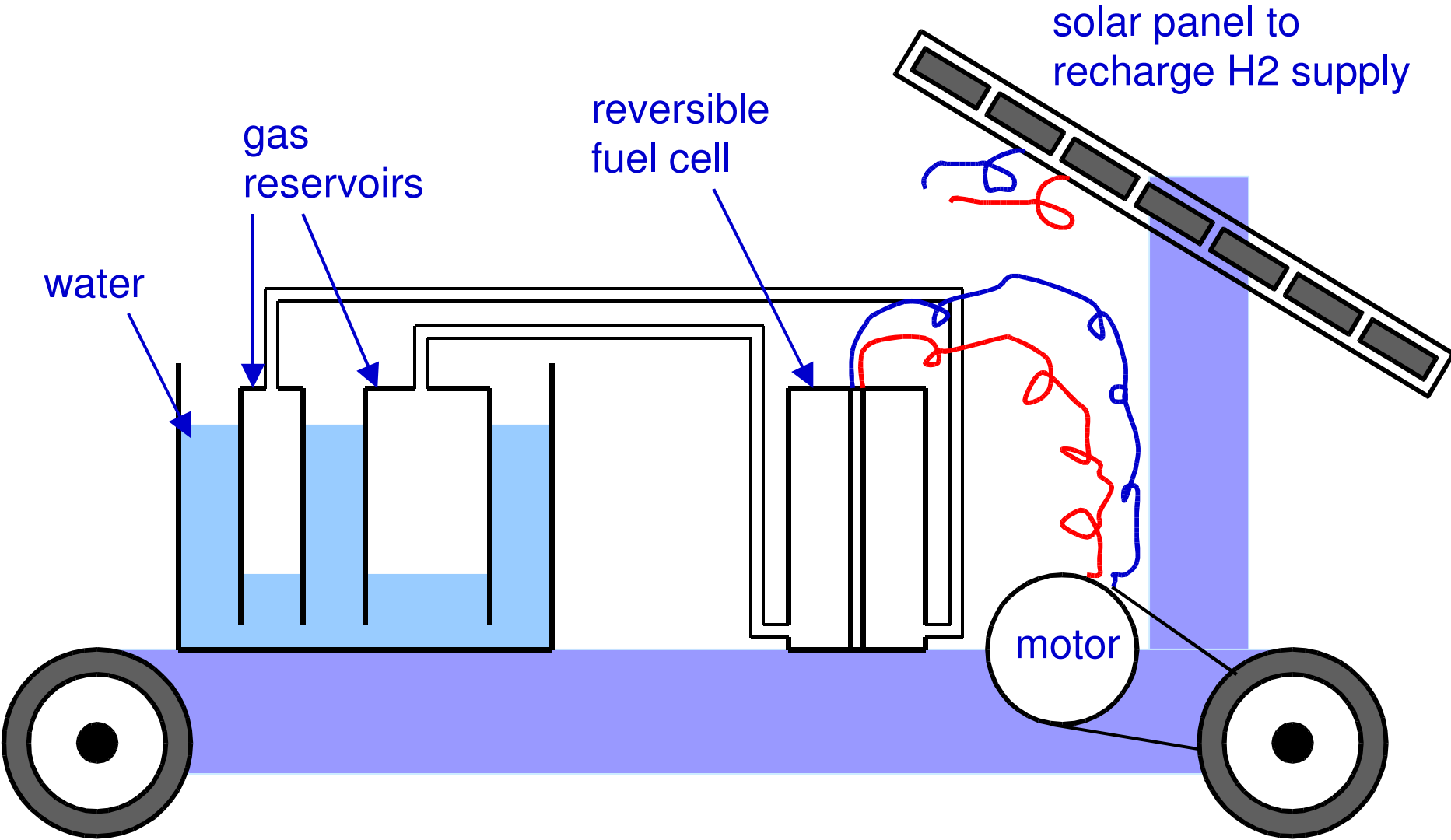


- **Hydrogen fuel cell** -> efficient and non-polluting
- **5000 psi hydrogen tank** -> 160 - 200 mile range
- **65 kW electric motor** -> 80mph+ top speed
- **High-voltage NiMH battery** as energy buffer (regenerative braking, acceleration boosting, startup power)
- **Small production run due 2004** for fleet trials -> not just prototype! (Prototypes running since March 2002)

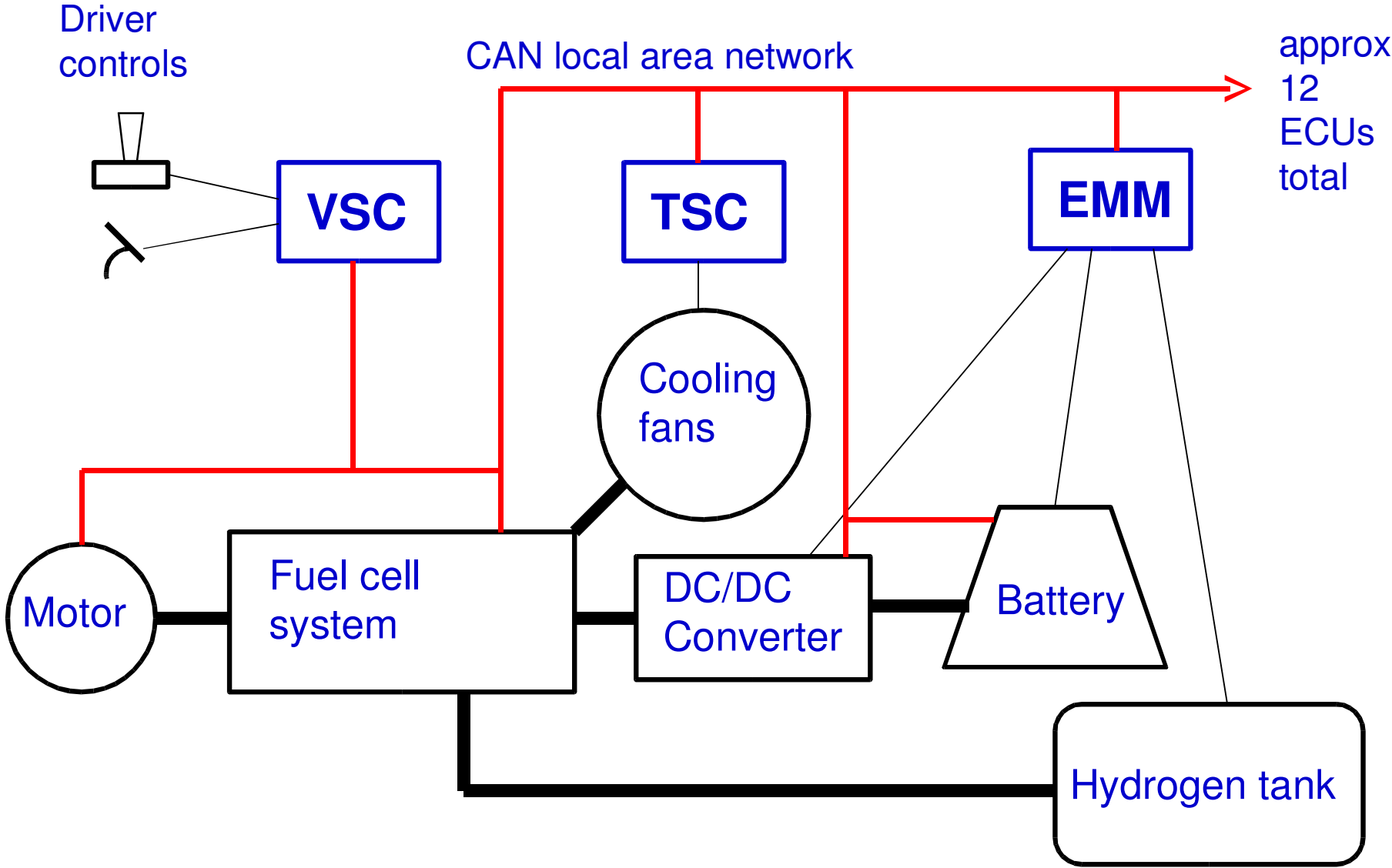
Hydrogen Fuel Cell Basics



Demonstration: model fuel cell car



Electronics: 3 new ECUs to develop



Our project: 3 brand new ECUs:

Vehicle System Controller (VSC)

Energy Management Module (EMM)

Thermal Systems Controller (TSC)

- **New** ECUs in **new** vehicle, so requirements expected to change as project proceeds
- Core hardware identical: Motorola MPC555-based powertrain ECU
- Production quality required, but low volumes

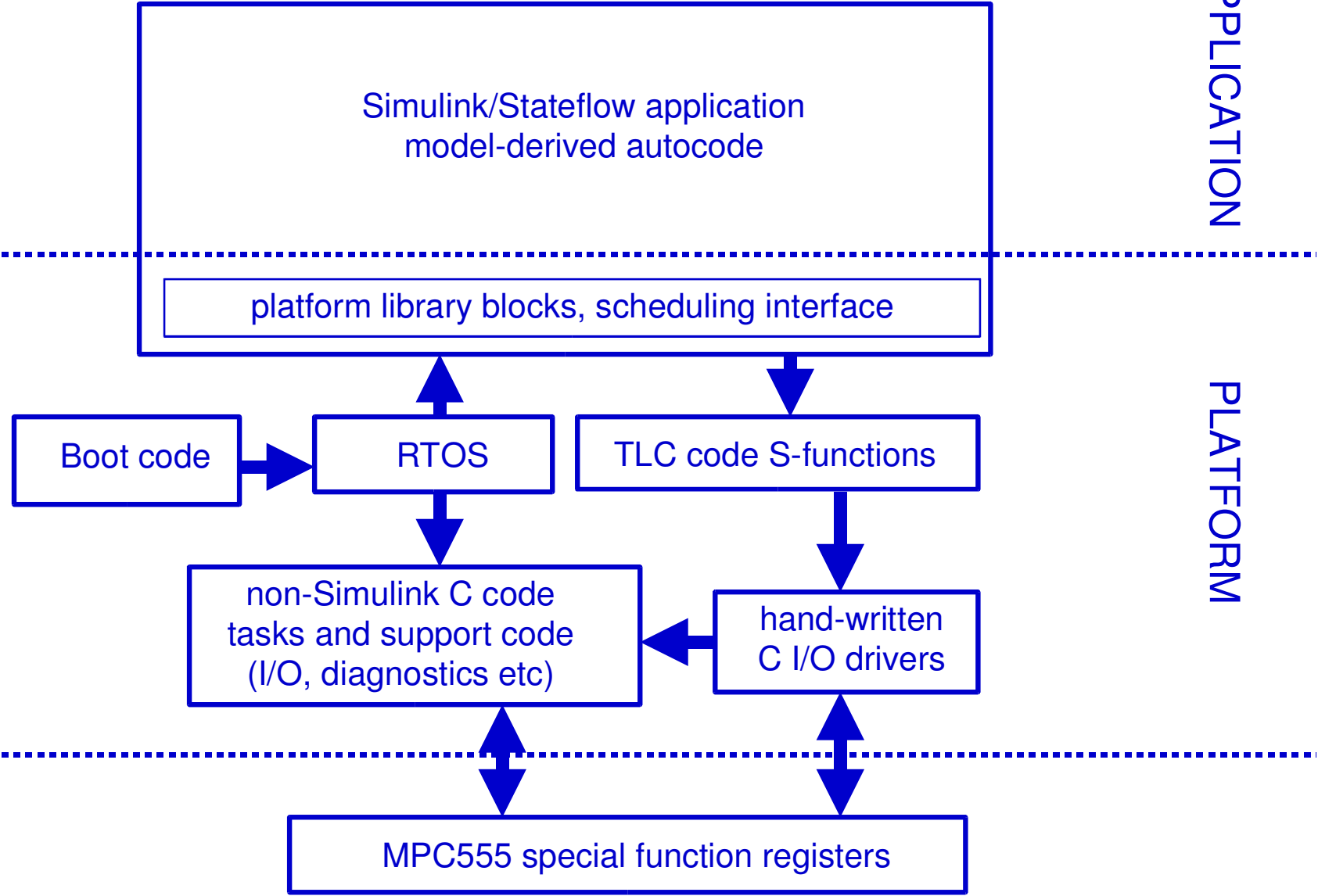
Auto-generated code?

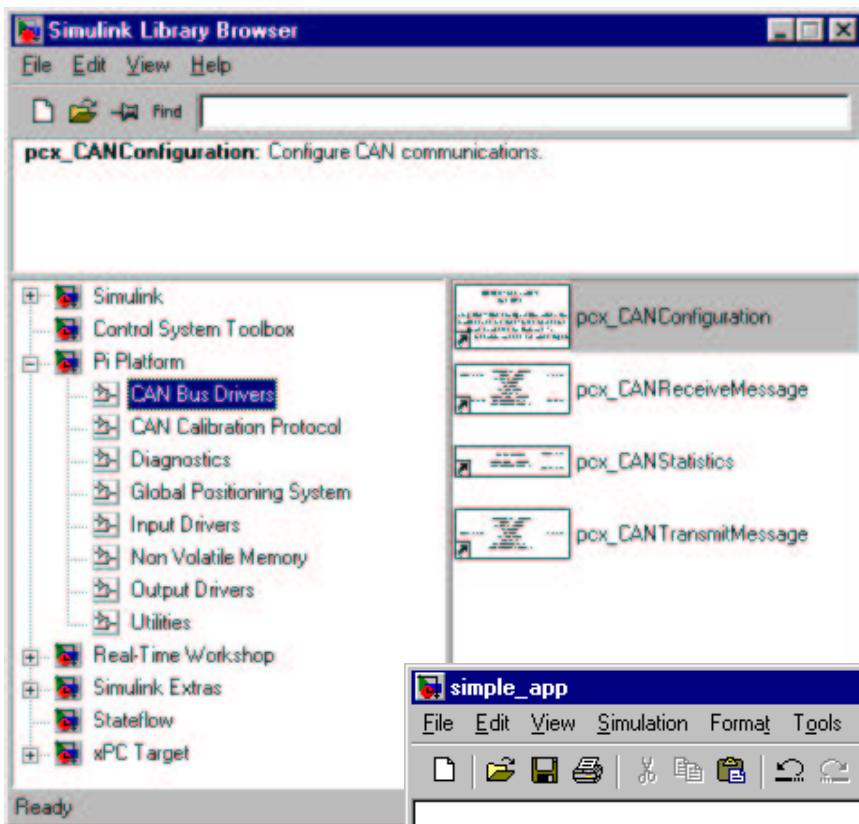
- Executable graphic MATLAB/Simulink designs
- Can run all or parts in simulation to iron out problems early
- "Autocode" generated directly by RTW
- Don't have to maintain matching hand code -- good here with a lot of expected change
- Don't introduce "human" bugs at coding stage
- Change design and rebuild for fast prototype
- Key: all done on actual production hardware

Autocode disadvantages

- RAM/ROM/CPU consumption may increase
- Lose control of code structure, style and naming (presents unit testing problems, calibration also affected)
- Significant upfront investment in producing reusable libraries for I/O, communications, diagnostics, scheduling...
- Locked in to particular vendor's tools (MathWorks)

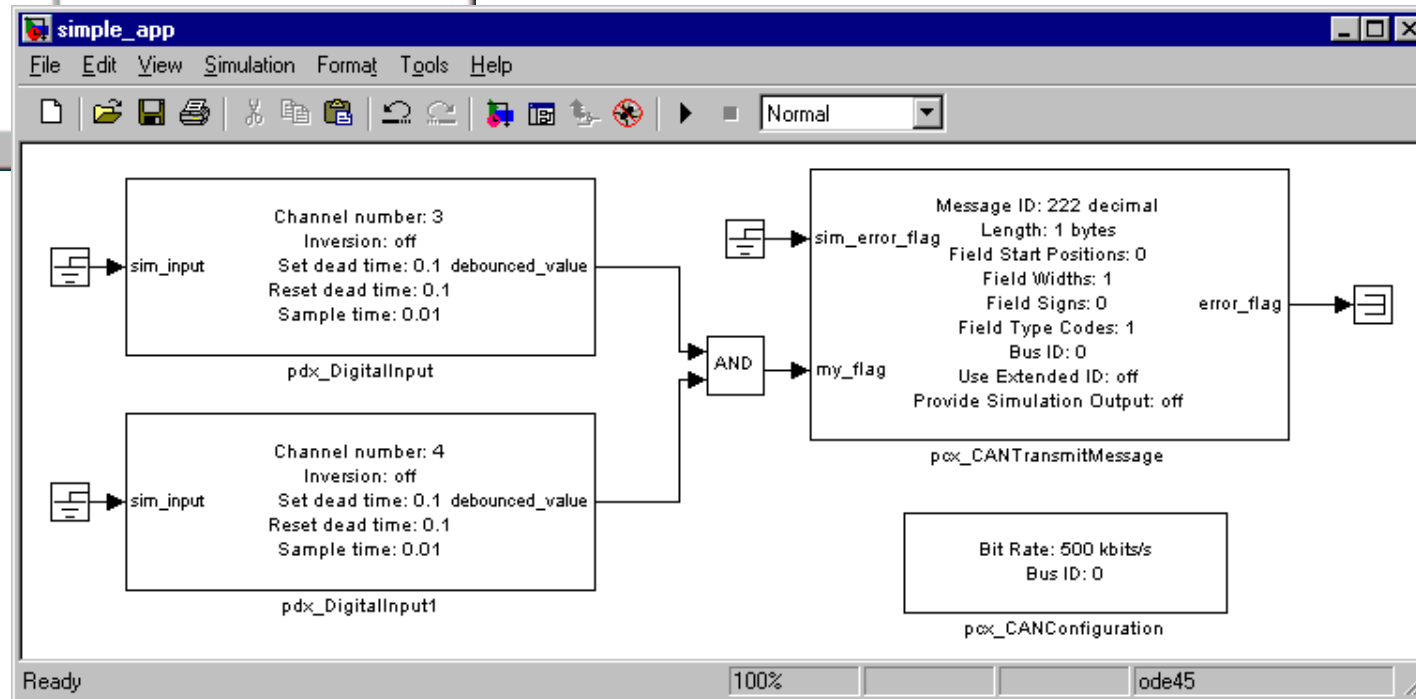
Software Architecture: Application and Platform



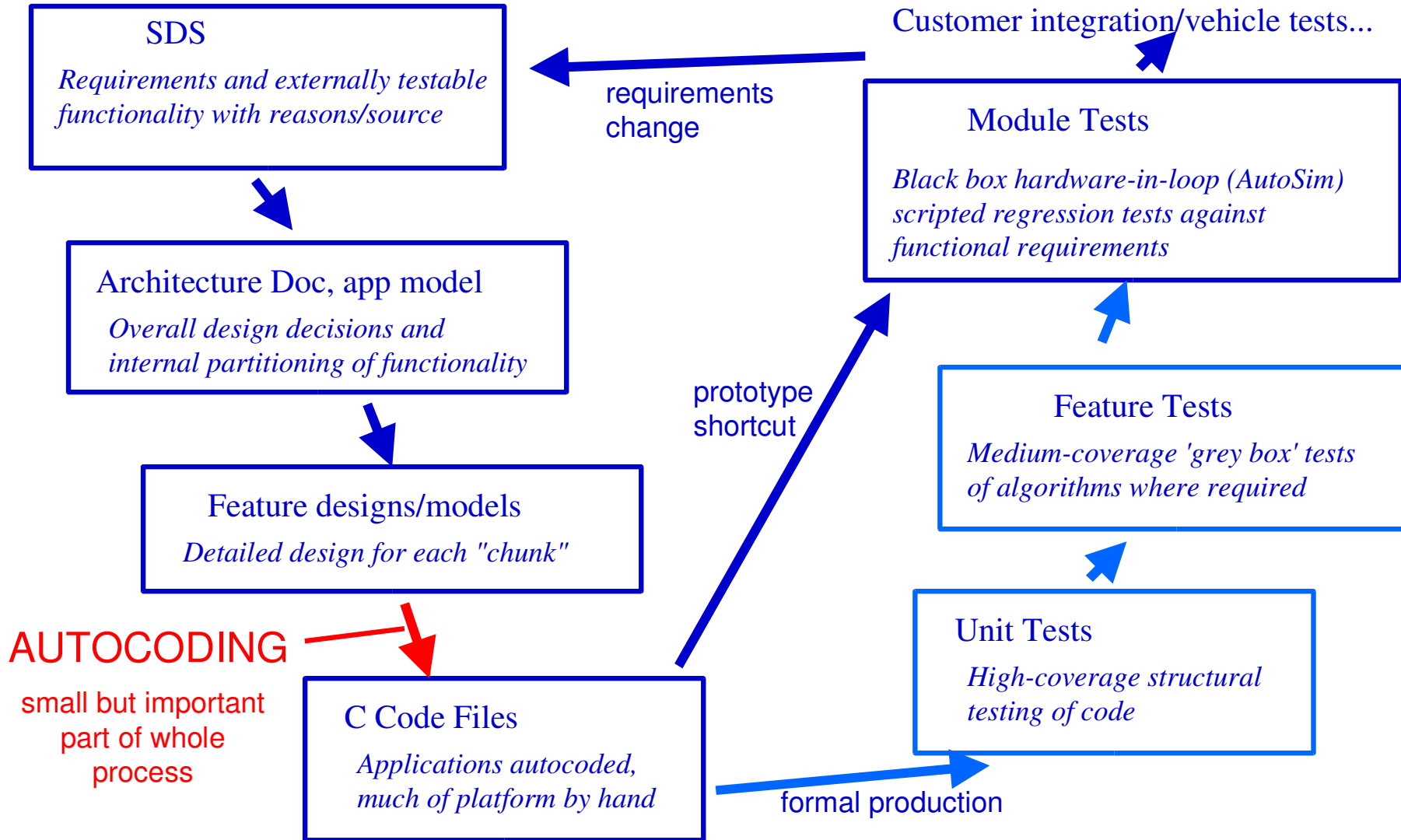


What it looks like in Simulink:

- Drag and drop custom library blocks to represent I/O channels, CAN messages etc
- Hit RTW 'build' button
- Download to target and run
- Just like prototyping system so thus far...



Context: Production Quality Development Process V-model



All steps: ISO 9001 quality plan, change control, version control, review, traceability, configuration management

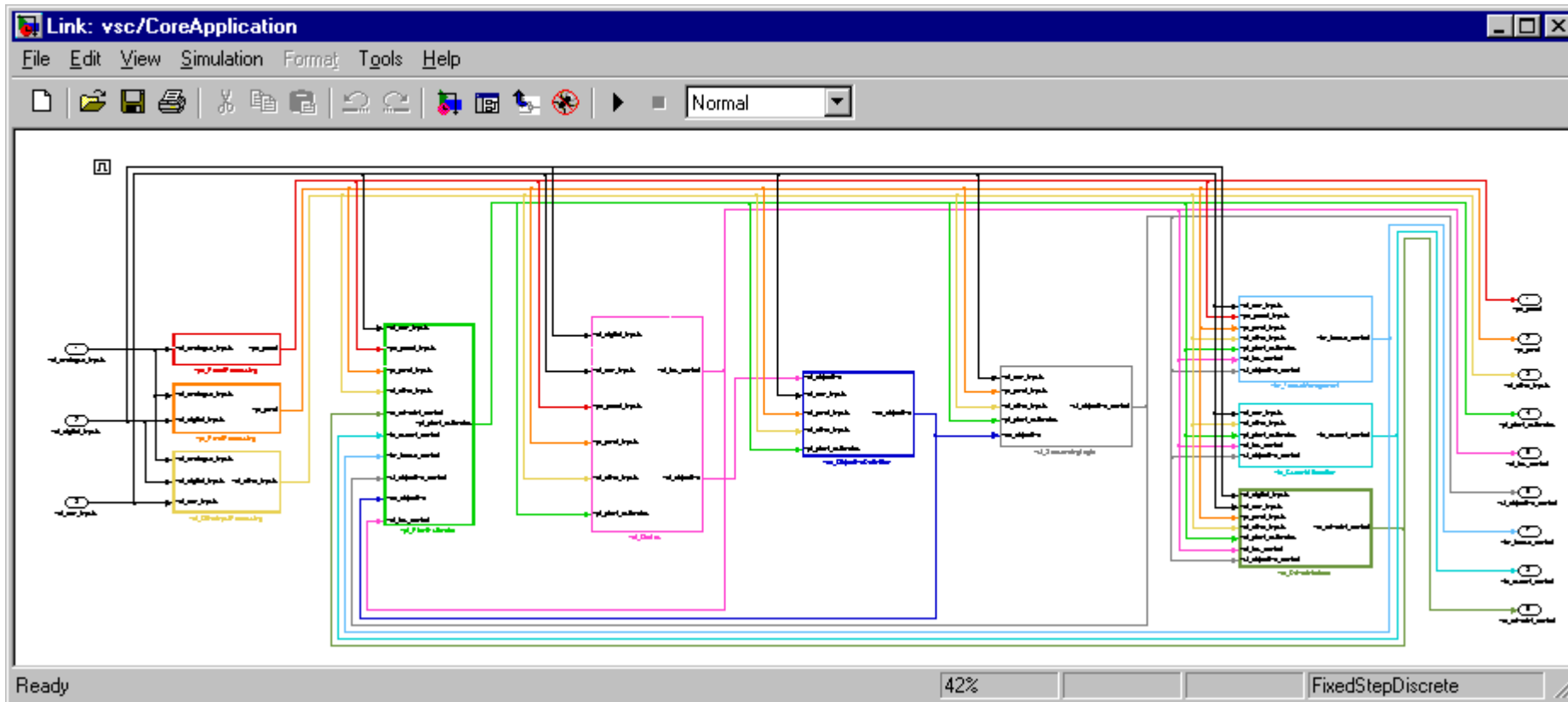
SDS (software requirements doc)

- Key specification for software and system tests
- Small, uniquely tagged requirements allow measurable traceability (Pixref tool)
- All testable from ECU external interface

- S(VSC IP 8) VSC shall set “VSC Status: PRNDL Position” equal to 8 (Drive) when PRNDL is in DRIVE position. (Ref 4 Section 4.1.1.5)
- S(VSC IP 9) VSC shall set “VSC Status: PRNDL Position” equal to 16 (Low) when PRNDL is in LOW GEAR position. (Ref 4 Section 4.1.1.5)
- S(VSC IP 38) VSC shall set “VSC Status: PRNDL Position” equal to 128 (Error) when there is a PRNDL Hard_Fault. (Ref 4 Appendix D, item 28)

Real Application Model

- Defines *feature* interface dataflows and can be built or simulated
- Each feature here just link to feature (library) model
- Separate files for formal team working and version control
- Some experimental changes, but too complex for ad hoc prototyping



Note: the detail of this figure is deliberately illegible; the control system details are confidential

Detailed Application Modelling

- As simple as possible, using own style guide
- Mainly Simulink, some Stateflow where appropriate
- Functional not time-based decomposition
- Simulink "wiring" shows data flow
- Requirements tag comments for traceability
- Separate data dictionary files detailing parameters and flows (units, range, etc)

What the Real-Time Workshop Autocode Looks Like

```
/* MinMax Block: <S107>/Maximum Integrator Clip */
{
  real_T max;

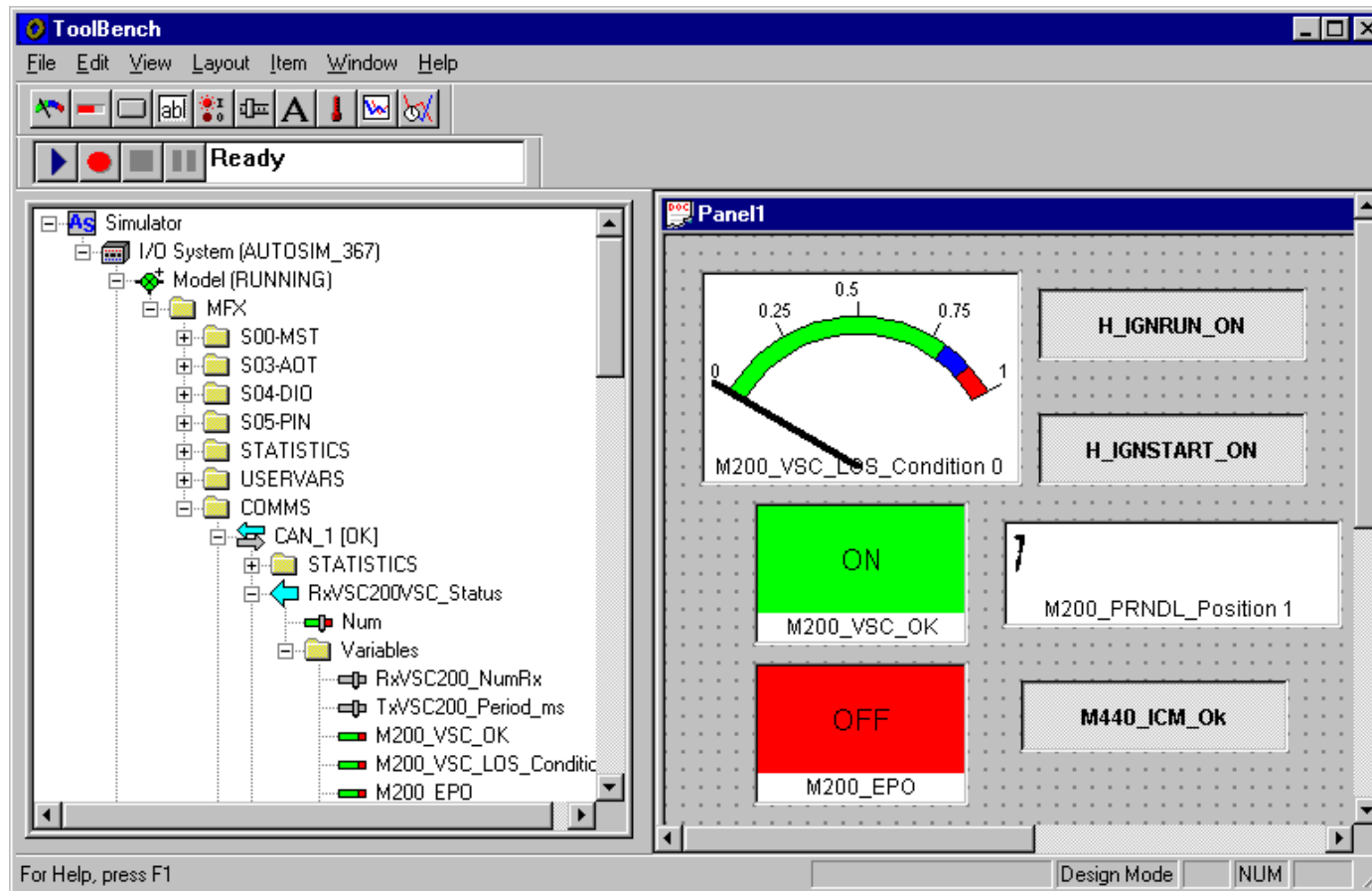
  max = rtB.temp299[0];
  if ((0.0) > max) {
    max = (0.0);
  }
  rtB.temp299[0] = max;
  max = rtB.temp299[1];
  if ((0.0) > max) {
    max = (0.0);
  }
  rtB.temp299[1] = max;
  max = rtB.temp299[2];
  if ((0.0) > max) {
    max = (0.0);
  }
  rtB.temp299[2] = max;
}

/* RelationalOperator Block: <S107>/Check Bucket Full */
rtB.s107_Check_Bucket_Full[0] = (rtB.temp299[0] >= (1.0));
rtB.s107_Check_Bucket_Full[1] = (rtB.temp299[1] >= (1.0));
rtB.s107_Check_Bucket_Full[2] = (rtB.temp299[2] >= (1.0));

/* Switch Block: <S107>/Switch Fault Flag Off? */
if (rtB.s107_Unit_Delay[0]) {
  rtB.s107_Switch_Fault_Flag_Off[0] = rtC_s108_Relational_Operator;
} else {
  rtB.s107_Switch_Fault_Flag_Off[0] = rtB.s107_Check_Bucket_Full[0];
}
if (rtB.s107_Unit_Delay[1]) {
  rtB.s107_Switch_Fault_Flag_Off[1] = rtC_s108_Relational_Operator;
} else {
  rtB.s107_Switch_Fault_Flag_Off[1] = rtB.s107_Check_Bucket_Full[1];
}
if (rtB.s107_Unit_Delay[2]) {
  rtB.s107_Switch_Fault_Flag_Off[2] = rtC_s108_Relational_Operator;
} else {
  rtB.s107_Switch_Fault_Flag_Off[2] = rtB.s107_Check_Bucket_Full[2];
}
```

Automated Hardware-in-the-Loop (HIL) System Testing

- Pi AutoSim provides CAN, digital and analogue I/O
- Fully scripted regression tests run automatically
- Scripts traceable to SDS, measurable functional coverage
- Automated testing of service diagnostics at same time



Code Performance: VSC (1300 CAN msg/sec)

| Release Version -> | V1.4 | V1.9 |
|---|---------|----------------|
| Code size | 232 KB | 330 KB |
| Calibrations and Constants | 23 KB | 26.5KB |
| TOTAL ROM REQUIREMENT | 250 KB | 356.5KB |
| Application/platform RAM | 11.5 KB | 18KB |
| Stack/RTOS RAM | 15 KB | 20KB |
| TOTAL RAM REQUIREMENT | 25.9 KB | 38KB |
| Approximate average CPU load | 30% | 45% |
| Approximate peak CPU load in worst 10 ms 'tick' | 50% | 60% |

Total RAM available: **58 KB**

Total ROM available: **1472 KB**

•Conclusion: autocode efficiency is adequate for target hardware here

Conclusions – Software Development

- Production autocode successful for these applications using formal development cycle
- Only one or two code generator bugs found in tests
- Invested in reusable *platform* for I/O and communications, written by hand (OpenECU); "prototype on production hardware" approach
- Improved software quality and response to requirements change, reduced effort
- RAM use probably increased but efficiency adequate for these applications
- But extra cost was involved in overcoming unit testing problems and creating autocode platform

Resources



Speaker: charlie.wartnaby@pитеchnology.com

Detailed SAE paper for this work:
2003-01-863 from <http://www.sae.org>

Pi Technology website: <http://www.pитеchnology.com>, see OpenECU and
<http://www.pитеchnology.com/downloads/files/FuelCells.pdf>

Ford website for this vehicle:
<http://www.ford.com/en/vehicles/specialtyVehicles/environmental/fuelCell/focusFCVHybrid.htm>
(Google: “Ford FCV hybrid”)

Model fuel cell car: <http://www.thamesandkosmos.com>