System Planning

Overcoming Gap Between Design at Electronic System Level (ESL) and Implementation

Horst Salzwedel, TU Ilmenau
Presented at EDACentrum Workshop: System Planning
Hannover, 30 November 2006

System Complexity

- chip complexity increasing (> 1 billion gates per chip)
- Synopsis of functionality to integrated chips (SoC)
- Components on chips become networked (NoC)
- Integration problems while put together ECUs (e.g. aircrafts up to 1000 ECUs, automobiles up to 100 ECUs)
- Exponential failure rates by extensive chips usage

  - Architectural complexity
  - Interdisciplinary functionality
  - Dynamic interactions between components

System Design Productivity

- Efficiency of tools grows with maximum 25% per year
- Complexity of electronic systems grows 65% per year
- Resolve system design gap by improving quality of specification

  - Abstract Modeling
    (gap between abstract model and implementation)
  - Executable specifications
    (validation and performance analysis)
  - Reuse of model components
    (functionality on different architectures)

Integration of requirements

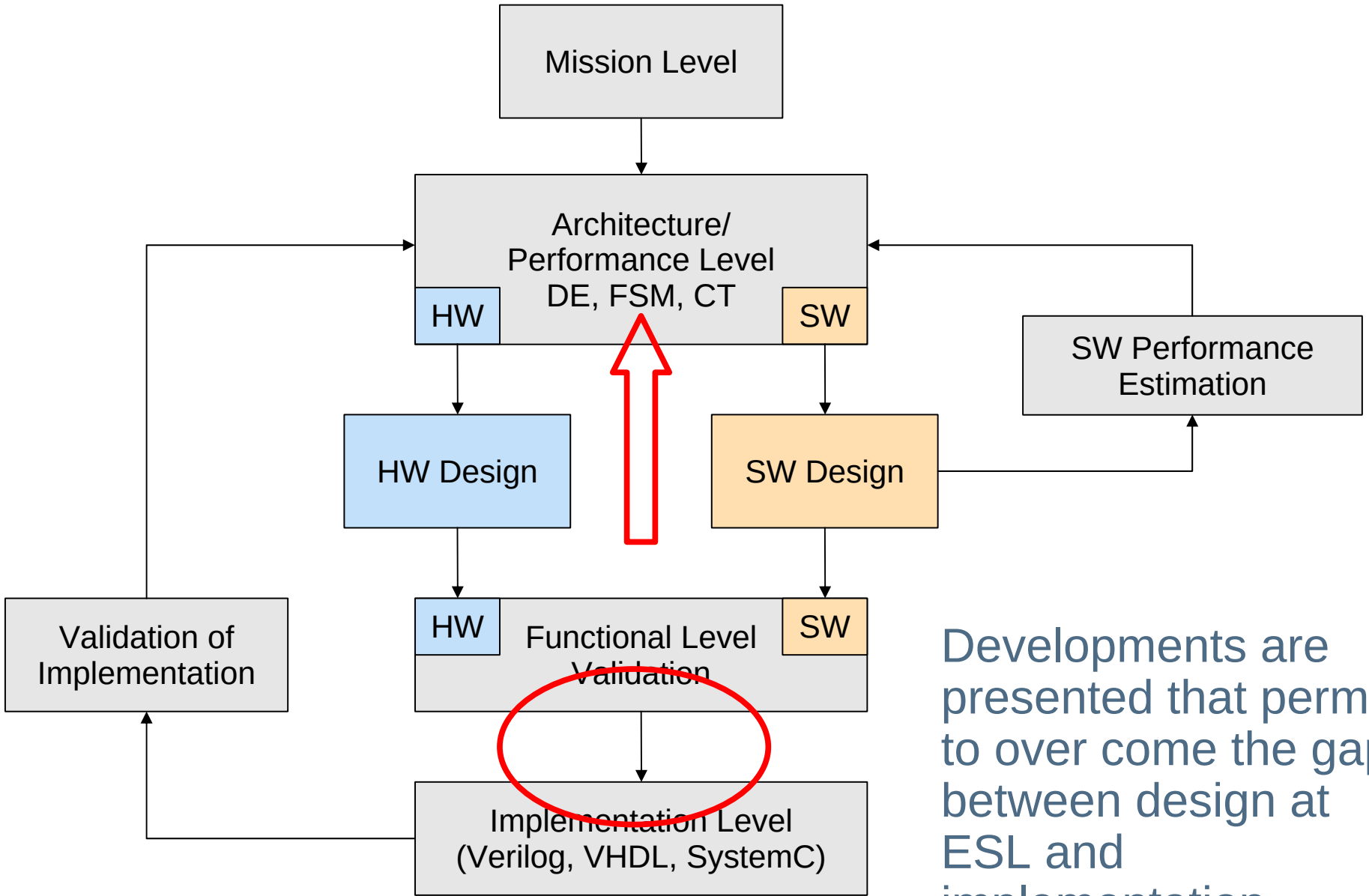
- Operational Requirements
  - Missions (use cases, test cases)
  - Physical environment
  - Functionality

    ➢ Modeling of system behaviour

- Performance Requirements
  - Shared and exclusive functional resources
  - Concurrent Processes
  - Transactions and workloads

    ➢ Modeling of system architecture

Significant progress has been made by moving many design iterations to Electronic System Level (ESL)



Developments are presented that permit to over come the gap between design at ESL and implementation

Mission Level Design Flow

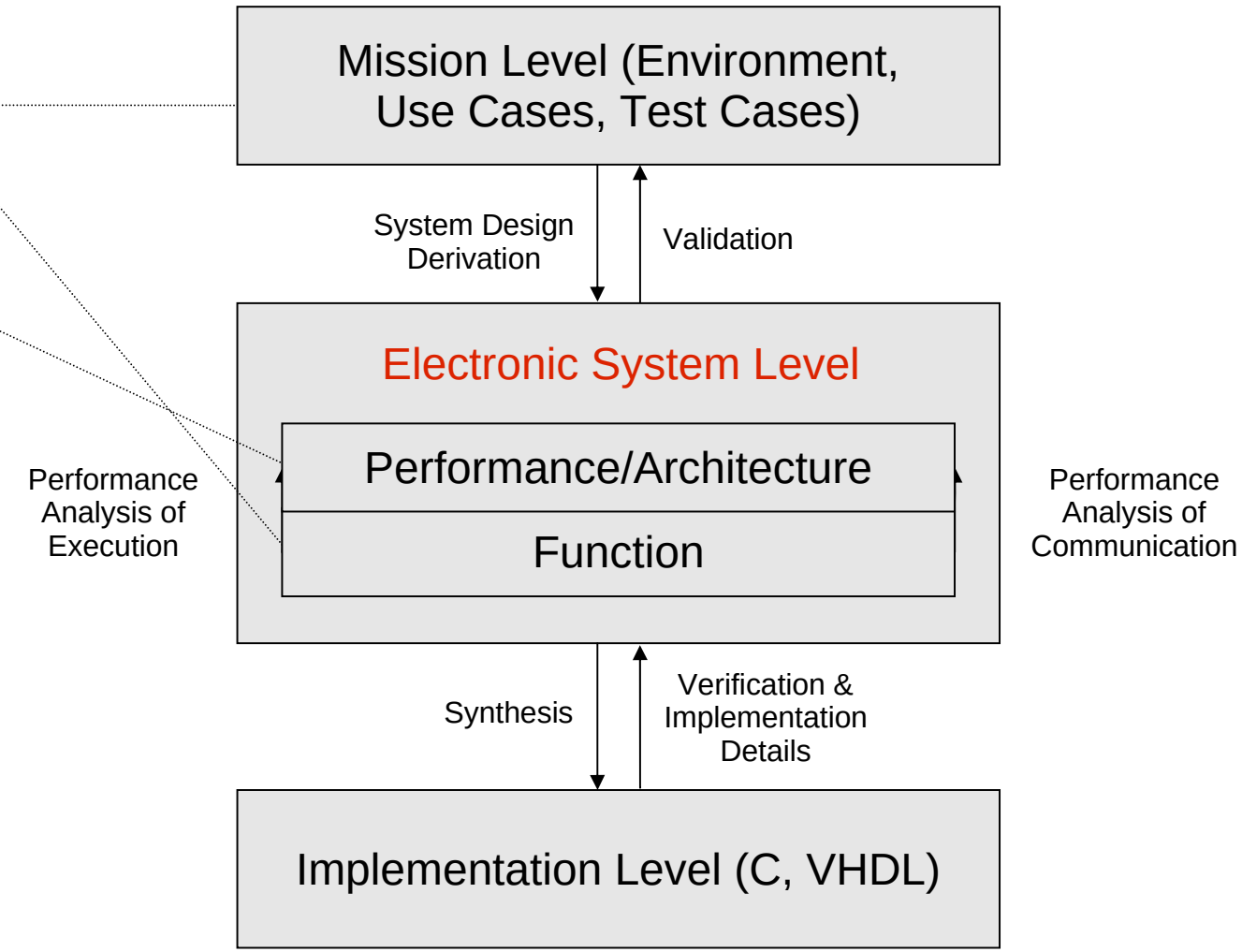
(1) Create Behaviour Model

(2) Create/Generate Architecture Model

(3) Performance analysis by simulation

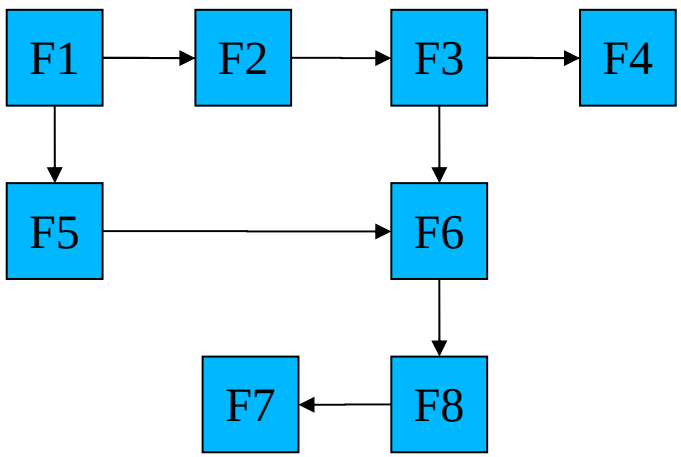
(4) Transform model into executable code

(5) Test and validate prototype



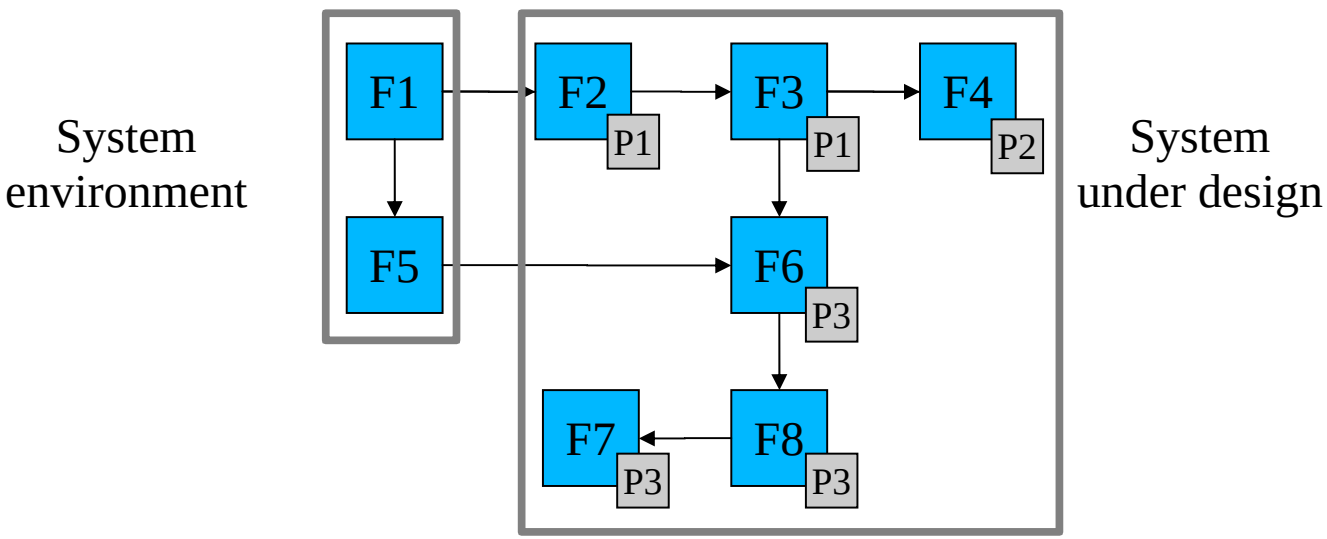
(1) Create Behavioural Model

- Specification of missions, environment and desired system functionality
- Concept for modeling:
  What should desired system do? ⟺ How to achieve?



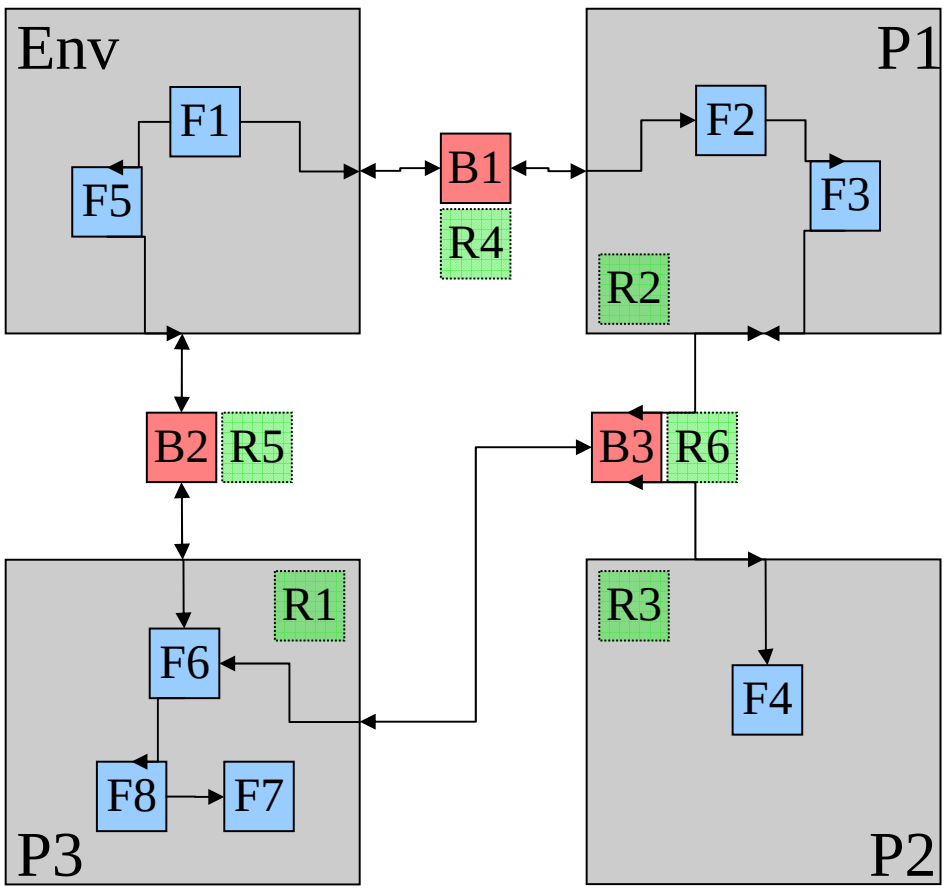
Create/Generate Architecture Model

- Definition of execution/communication architecture (allocation)
- Coupling of behaviour and architecture (binding)
  - Semi automatic: generation of default Architecture Model based on annotated Behaviour Model
  - Manually: Modelling of Architecture Model (usage of library model elements) and mapping of components from Behaviour Model



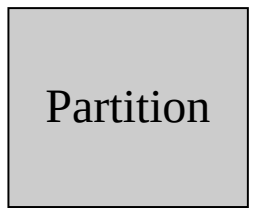
Architecture Model - Features

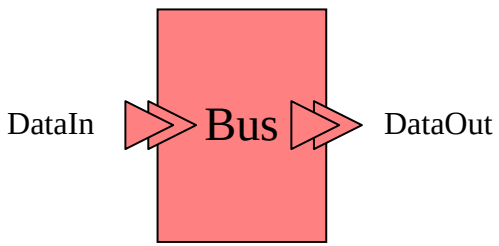
- Definition of execution and transaction architecture
- Iteration over architecture (generic bus interfaces)
- Utilize XML standard and tools (XSLT)
- Performance analysis by model execution (DE)
- Starting point for synthesis
- Electronic System Level (Functional Level and Performance/Architecture Level together)

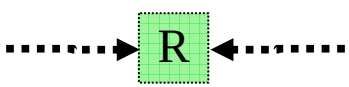


Architecture Model - Elements

- Partitions
  - Modeling of execution components as functional resources (mapping of functions)
  - Determination of execution units for synthesis
- Abstract extensible buses
  - Modeling of communication components (passive, active) to connect partitions (n to n)
  - Mapping of functional data flows onto architectural data flows
  - Internal generic interfaces and usage of standardized data structures
  - Performance and interconnection schemes
  - Determination of bus types for synthesis
- Linkable resources
  - Representation of abstract bus and partition resources by queues and servers





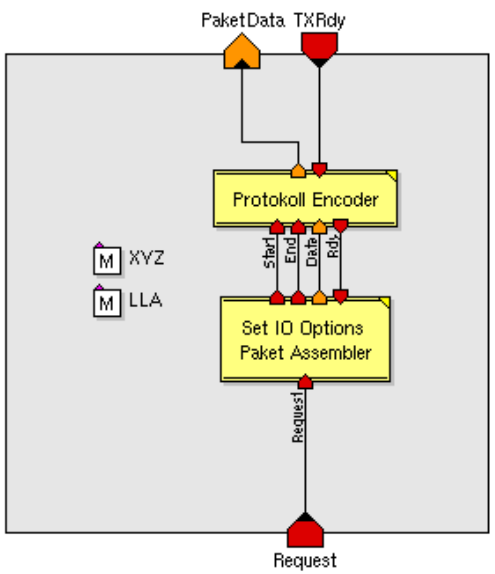


(3) Performance analysis by simulation

- Processing amount and structure of informations, degree of attainable parallelism and necessary performance
- Analysis of time-related Architecture Model informations
  - Queue size of interfaces
  - Bus usage rates
- Refinement of execution and communication architecture
  - Parametrization and replacement of buses (Wishbone, EIA232, USB 2.0, FlexRay, Zigbeex)
  - Parametrization of interface queues
  - Modification of resource linking states
- Usage of existing buses (Network Block Set) or definition of new buses (pattern approach)

(4) Transform model into executable code

- Transformation into HW/SW prototype code (VHDL, C) and documentation (utilize XSLT standard and tools)
- Controlled and fine tuned by preliminary annotations within the model
- Flexible management of changes within the model and the generated code

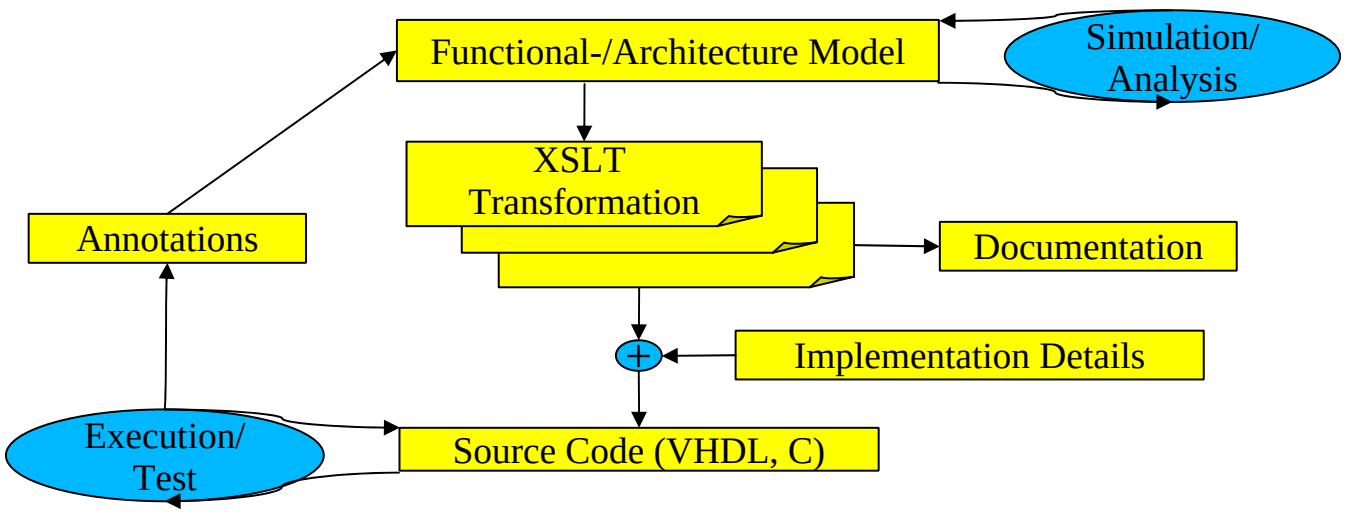


Example

```
-- Implementation of module TSIPCommandModule
-- component instantiation
i_ProtocolEncoder: ProtocolEncoder
  port map(
    Start => Relation3,
    Ende => Relation2,
    Data => Relation1,
    PaketData => Relation5,
    TXRdy => Relation4,
    Rdy => Relation6 );

i_SetIOOptions: SetIOOptions
  port map(
    Request => Relation7,
    Data => Relation1,
    Start => Relation3,
    Ende => Relation2,
    TXRdy => Relation6 );

-- external component connections
PaketData_reg <= Relation5;
Relation7 <= Request_reg;
Relation4 <= TXRdy;
```

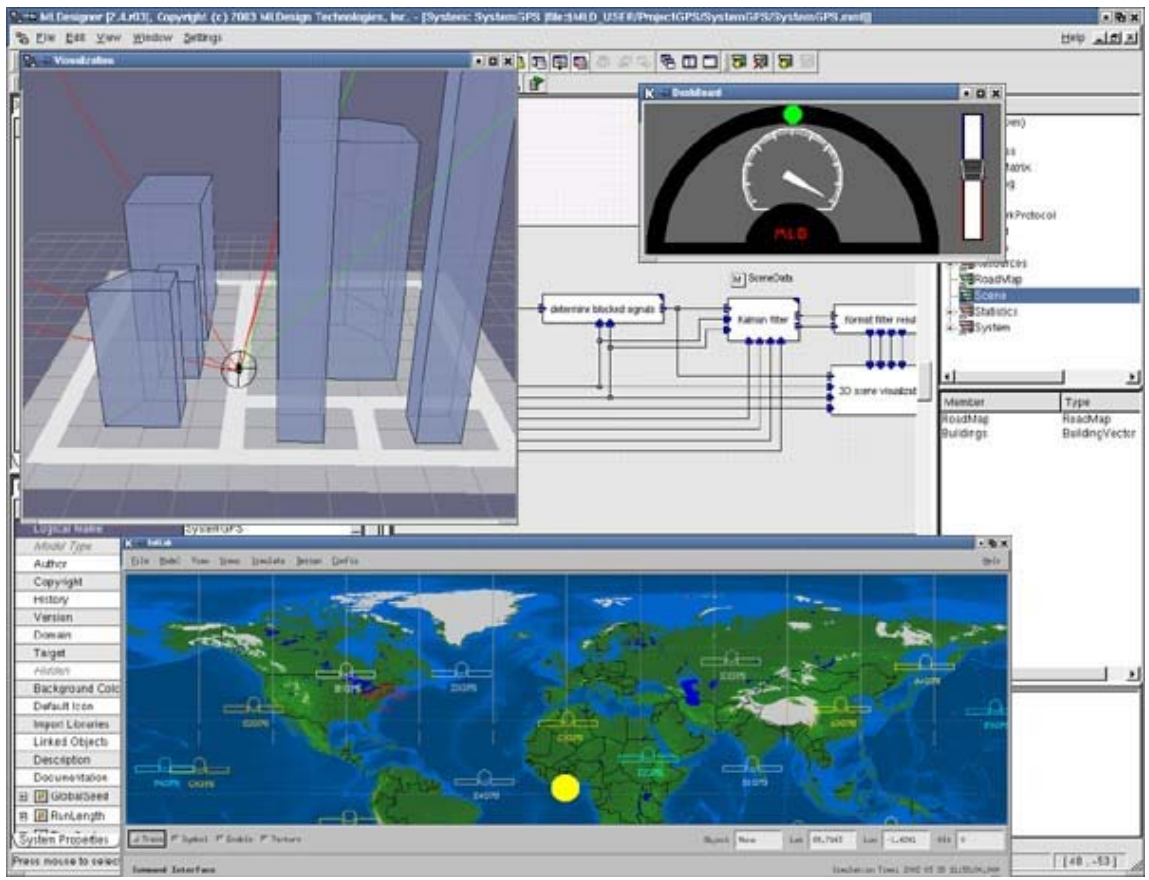
(5) Test and validate prototype

- Easy to refine prototype by changes within the model or the generated code
- Compare performance results of model and the generated code
- Semiautomatic transformation allows tracking of requirements through stages of development

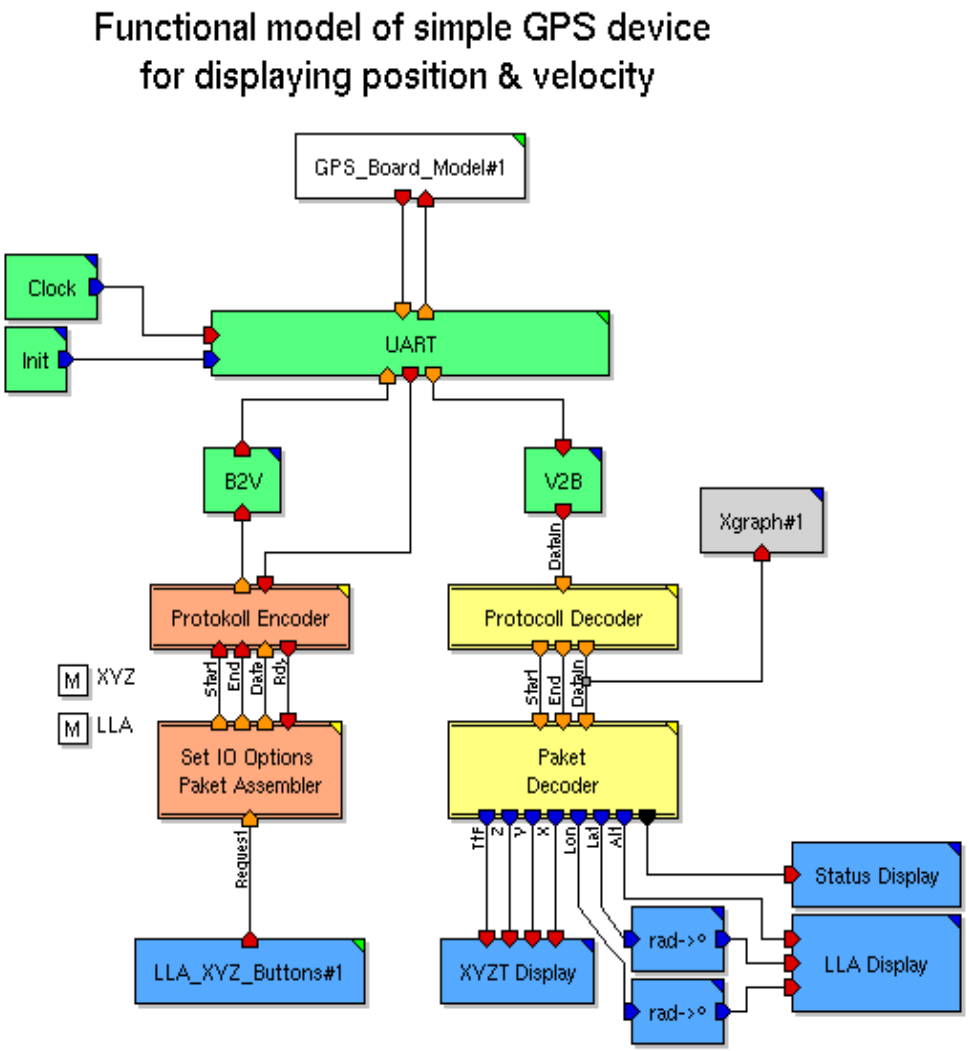


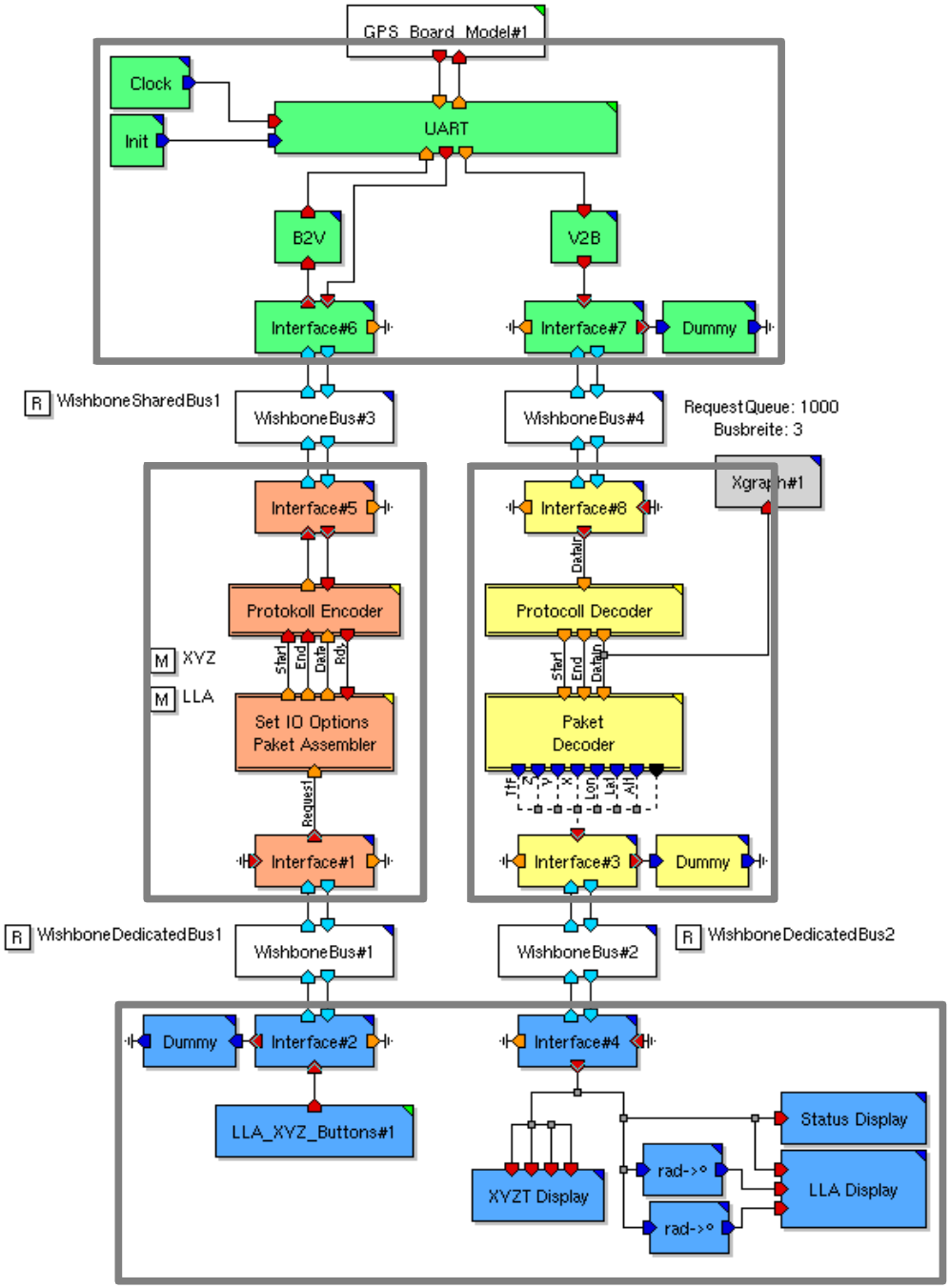
Virtual SatNav Validation Environment

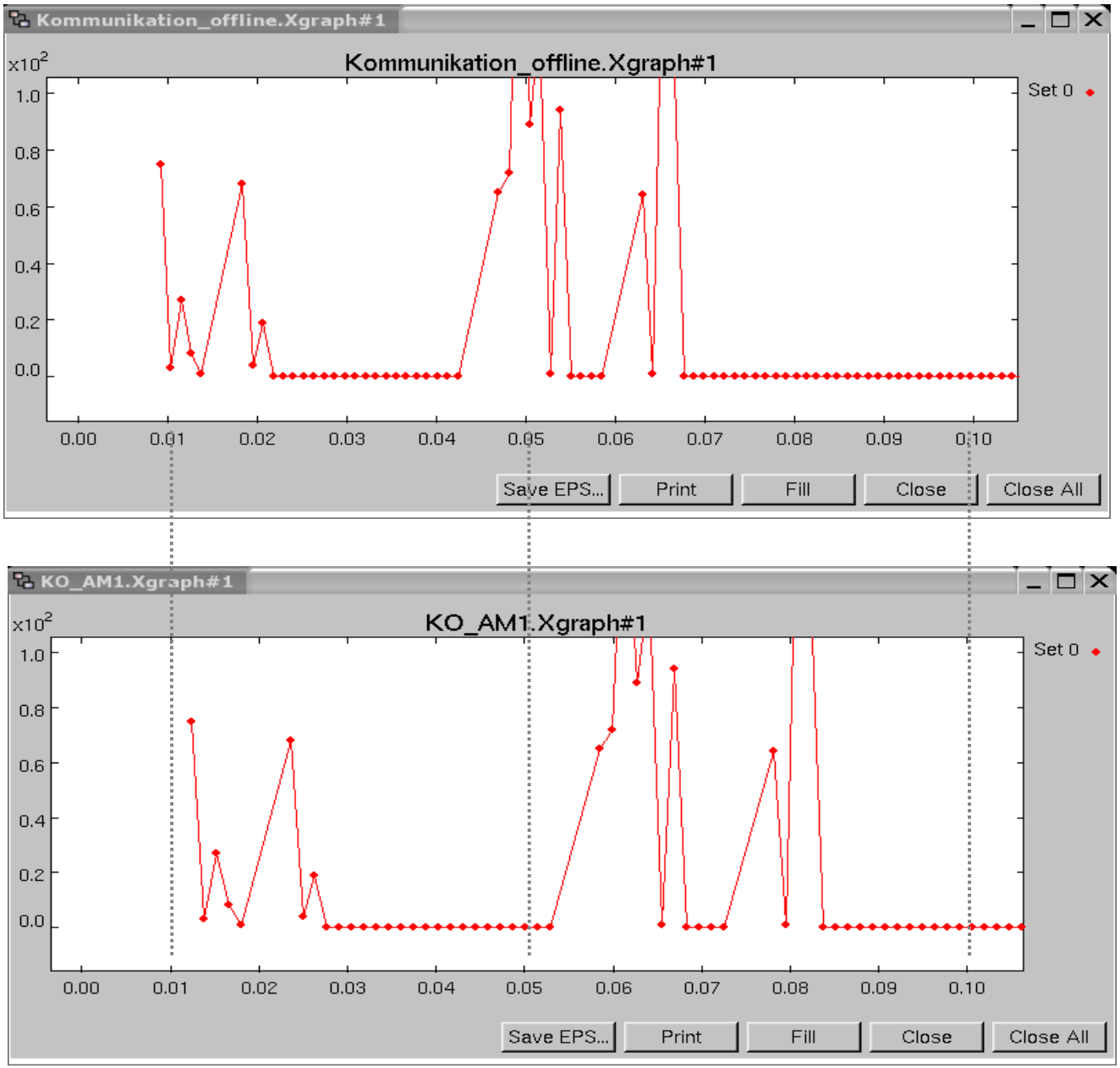
- navigation system model (MLDesigner)
- navigation satellite model (SatLab)
- data driven 3D-scene (OpenGL)
- car controller widget (Tcl/Tk)



Example: GPS Receiver Interface





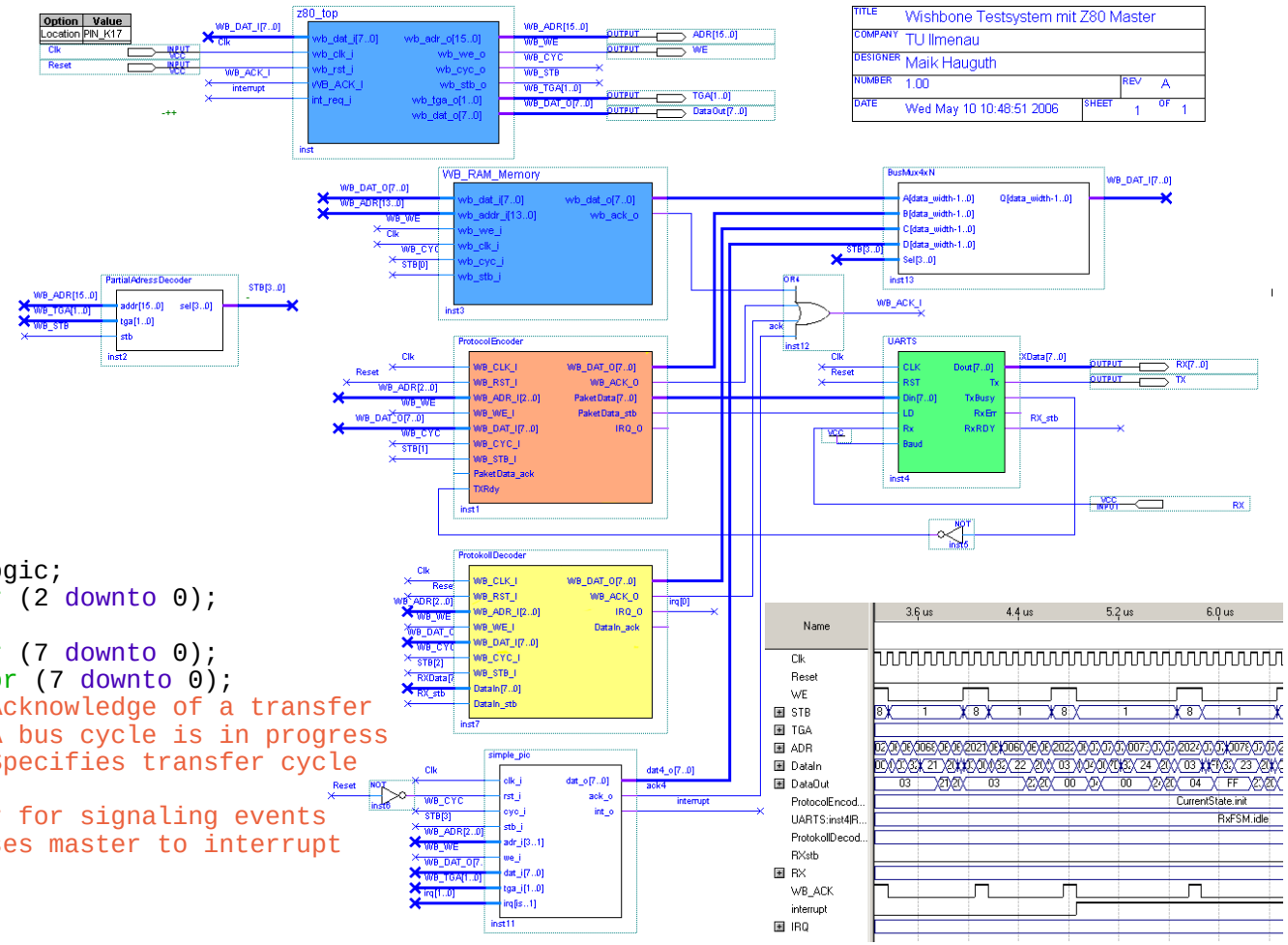


Without Architecture Model

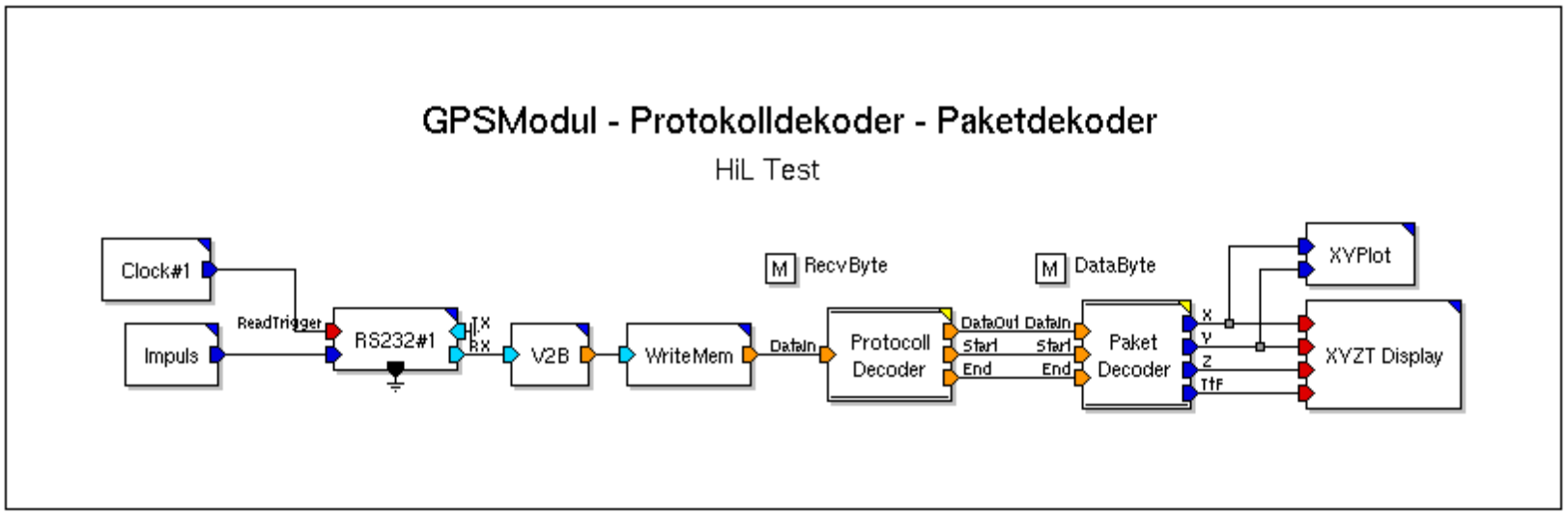
With Architecture Model
(bus specific delays)

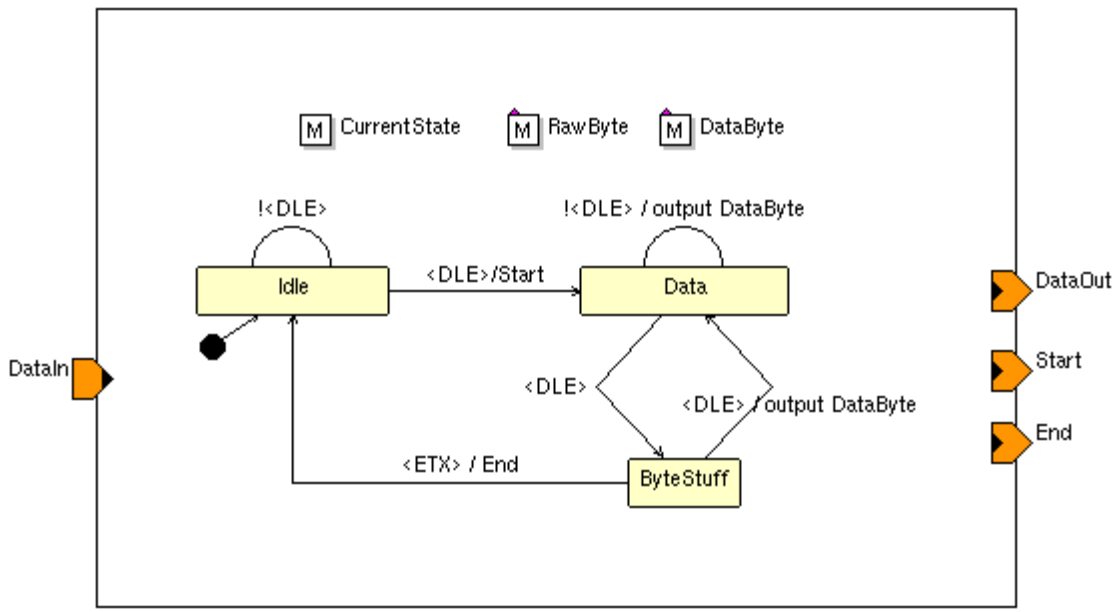
FPGA Implementation

```
-- IP Core ProtocolEncoder
--
-- Conforms to Wishbone SoC

library ieee;
use ieee.std_logic_1164.all;

entity ProtocolEncoder is
  port (
    -- WISHBONE ports
    WB_CLK_I, WB_RST_I: in std_logic;
    WB_ADR_I: in std_logic_vector (2 downto 0);
    WB_WE_I:  in std_logic;
    WB_DAT_I: in std_logic_vector (7 downto 0);
    WB_DAT_O: out std_logic_vector (7 downto 0);
    WB_ACK_O: out std_logic; -- Acknowledge of a transfer
    WB_CYC_I: in  std_logic; -- A bus cycle is in progress
    WB_STB_I: in  std_logic; -- Specifies transfer cycle

    -- Interrupt request register for signaling events
    IRQ_O: out std_logic; -- causes master to interrupt
      ...
```



GPS Development: HW in the loop





Summary

- A methodology was presented to design at ESL and map the design into an implementation, consisting of
  - functional design with virtual GPS development system
  - automatically mapping of plug and play HW models into functional design
  - iterating on integrated HW/SW model until functional requirements are met with the selected HW
  - semi-automatically mapping selected partitions into VHDL -> FPGA
  - hardware in the loop simulation with FPGA
- The methodology was validated for an example of a GPS based navigation system