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

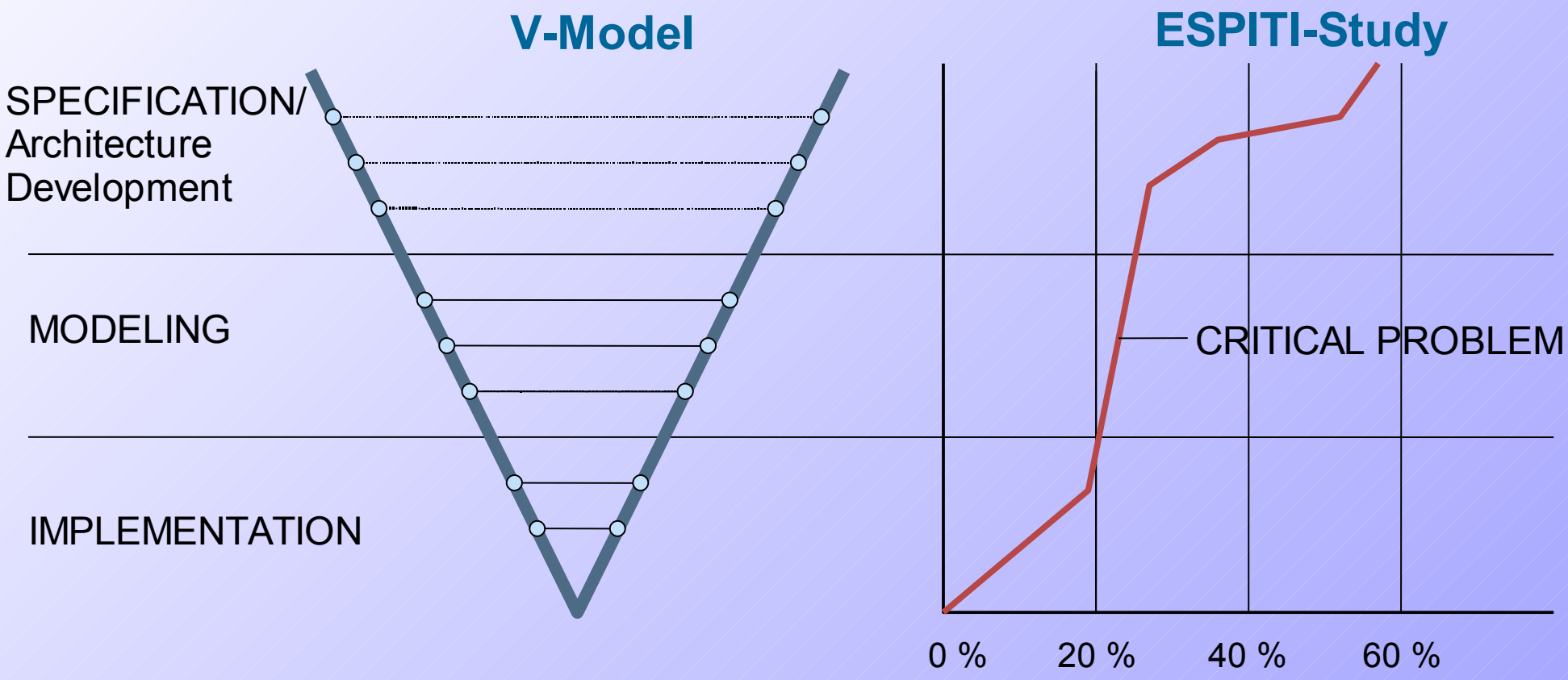
Tommy Baumann, Maik Hauguth, Horst Salzwedel
Technical University of Ilmenau
WMC'07, San Diego, January 15th, 2007
host.salzwedel@tu-ilmenau.de

Model based design techniques at functional level have been used in aeronautics for more than 100 years

- 1980th Design challenges:
  - Interdisciplinary design of aircraft stability augmentation systems with different tools and different design methodologies caused problems in all aircraft prototypes
  - Integrated flight propulsion control introduced major communication problems
  - Gap between design and implementation
- Solutions
  - Development of common description languages and tools, e.g., CtrlC/ModelC, MatrixX/SystemBuild, later: Matlab/Simulink
  - Independent verification of specification and implementation

1990th Design Challenge: Networked avionics with more than 100 electronic control units (ECUs) (similar to situation in automobiles today)

Failure rate in HW/SW designs



Solution:

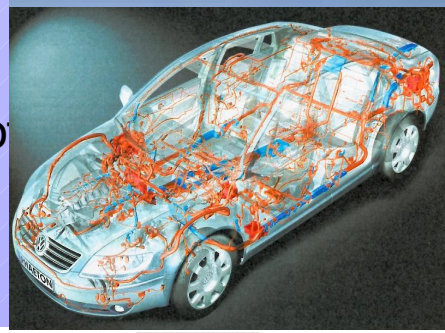
- Development of executable specifications of embedded flight control systems at performance level using event based simulator
- Moving chip design from logical level to functional level

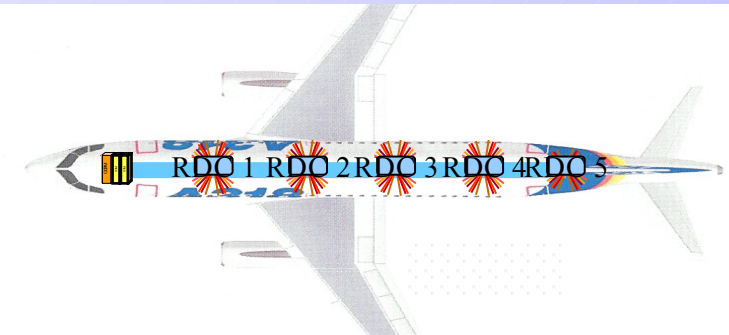
Design Challenge 2000:

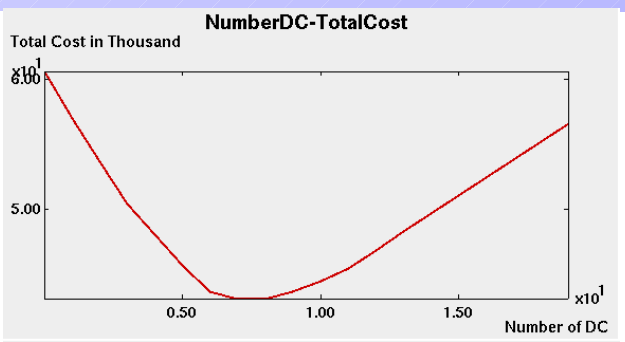
- chip complexity increasing (> 1 billion gates per chip)
- Components on chips become networked (NoC)
- Aircraft have 1800 and more dynamically interacting ECUs (automobiles up to100)
- Management cost have risen to more than 35% (for complex chip design)

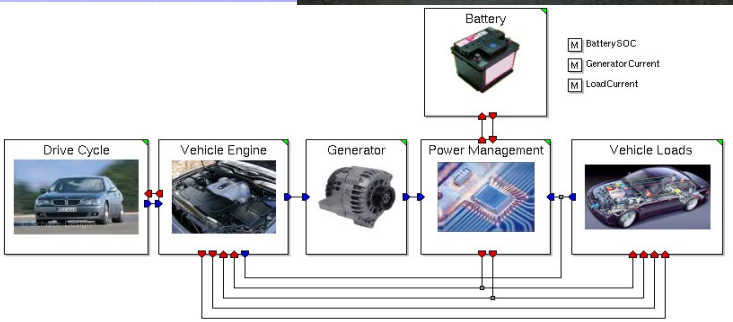
Solutions:

- Integrated design methodologies from mission level to implementation, use of multi-domain development tools: Speedup of simulation schedulers 100x (DARPA TTNT, NavAir WDM projects)
- Formal executable specification: Reduction of simulation memory models >10x (NavAir MMA project, automotive power management)
- Modeling and optimization of organization and design flow: speedup o design 2x-10x (Automotive electronics development)
- Optimization and automated generation of models for networks in aircraft









Goal of this research

In order to cope with the expoentially rising complexity in design of networked embedded systems, the level of abstraction in design had to be constantly raised

The result is a widening of the gap between design and implementation

The goal of this research is to develop a methodology to overcome the gap between design and implementation

Methodology: ESL to Implemenation

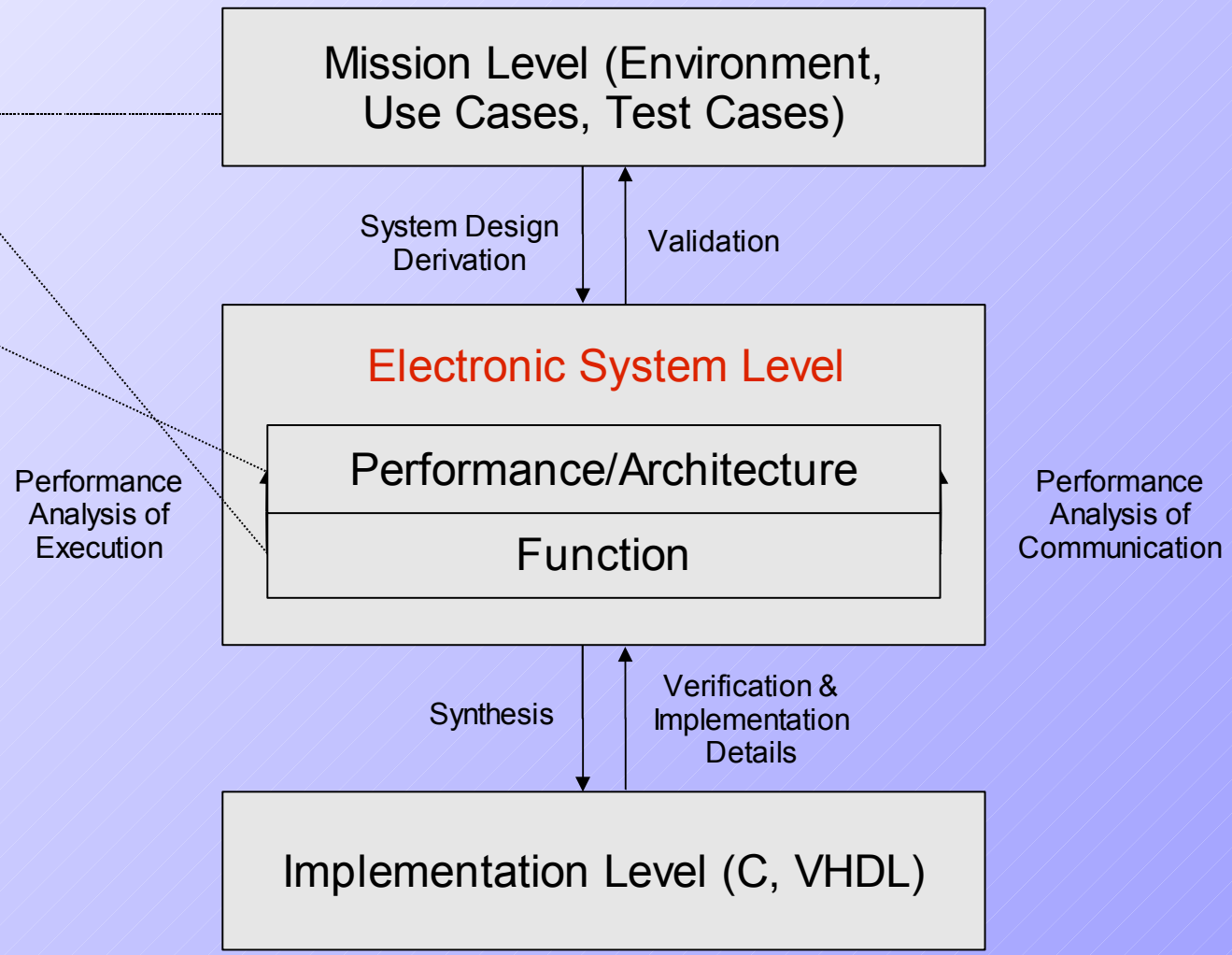
(1) Create Behaviour Model

(2) Create/Generate Architecture Model

(3) Performance analysis

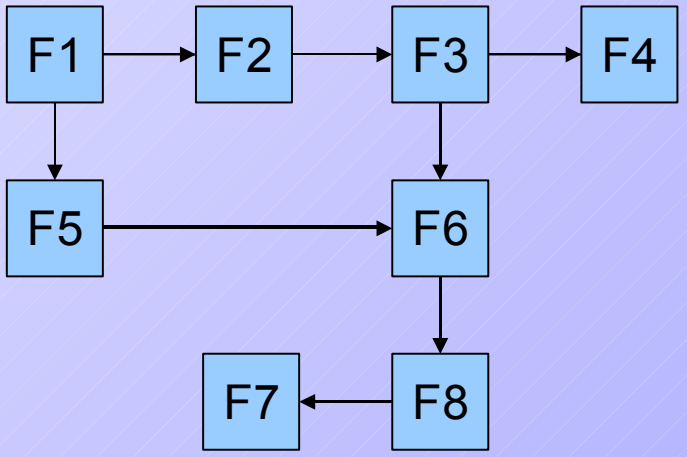
(4) Synthesize model into implementation

(5) Test and validate prototype



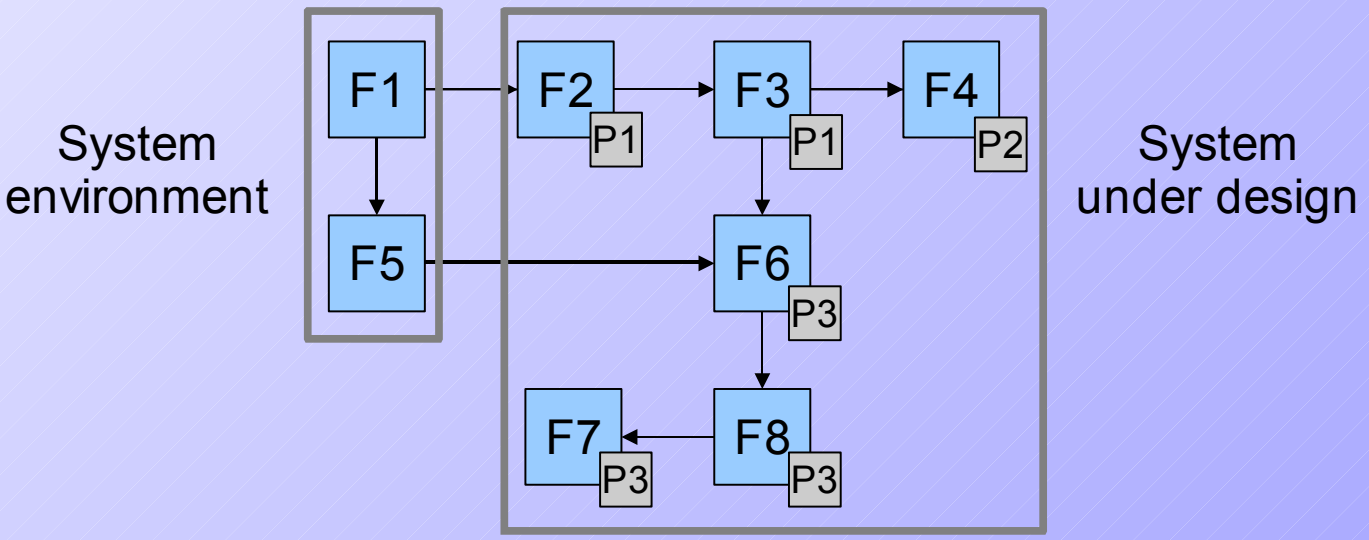
Step 1: Create Behavioral Model

- Specification of missions, environment and desired system functionality
- Translate concept into model: ⟺
  What should desired system do?



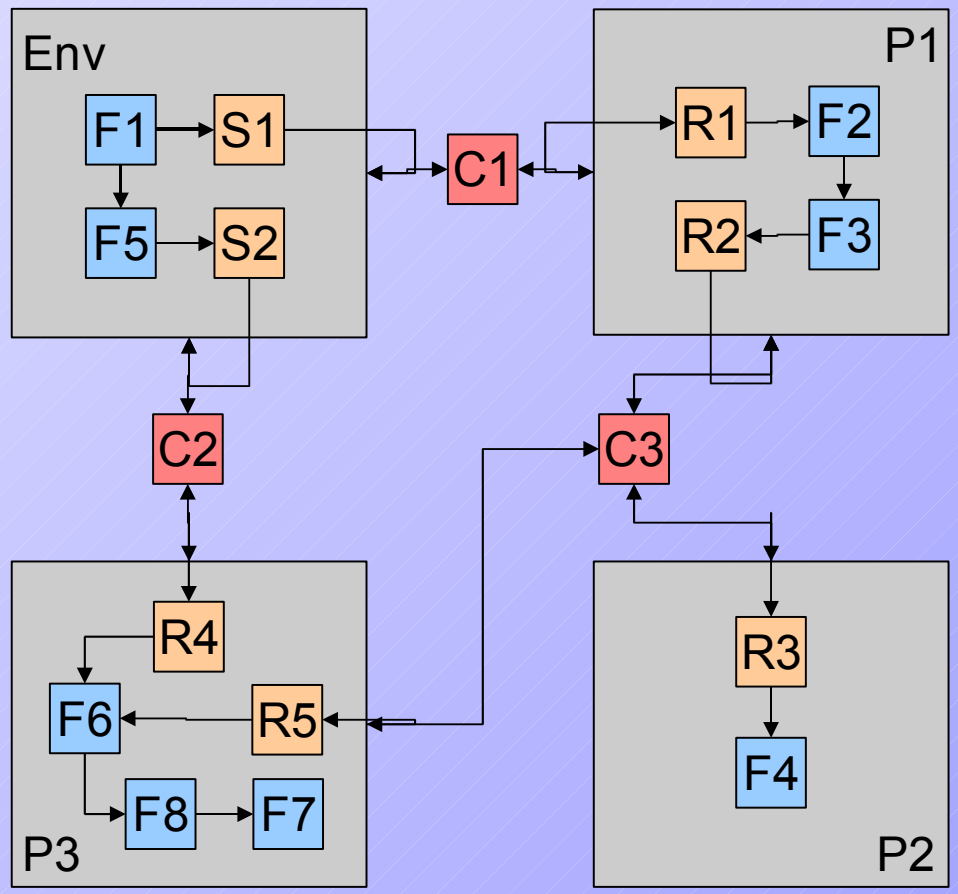
Step 2: Generate/Create Architecture Model

- Semi Automatic: generation of different Architecture Models based on annotated Behaviour Model by the BM2AMGenerator
- Manually: design Architecture Model directly with Network Block Set and Behavioural Model elements



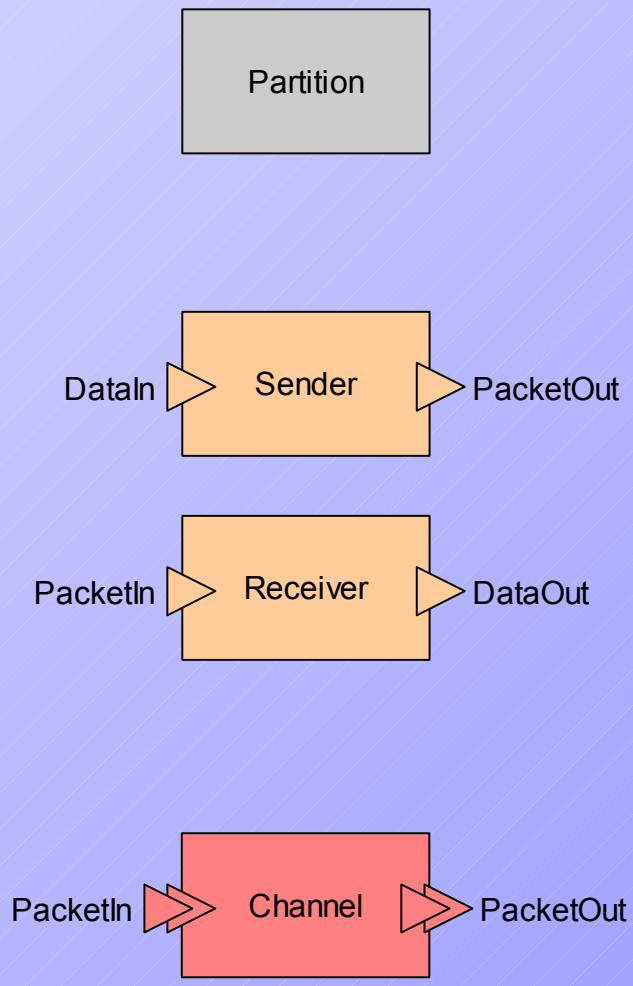
Architecture Model - Features

- Definition of execution and communication architecture (allocation)
- Coupling of behaviour and architecture (binding)
- Generic interfaces for fast iteration over architecture (Network Block Set)
- Performance analysis by model execution (DE)
- Starting point for synthesis
- Electronic System Level (Functional Level and Performance/Architecture Level together)



Architecture Model - Elements

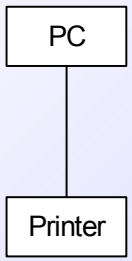
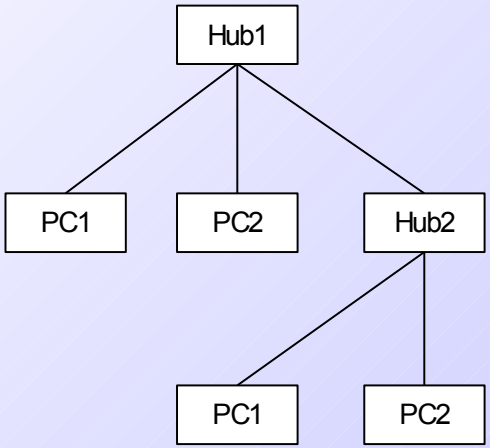
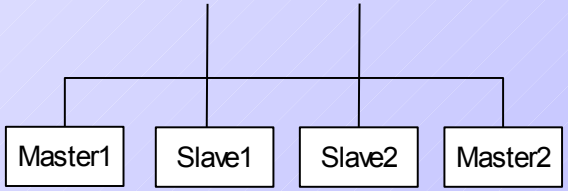
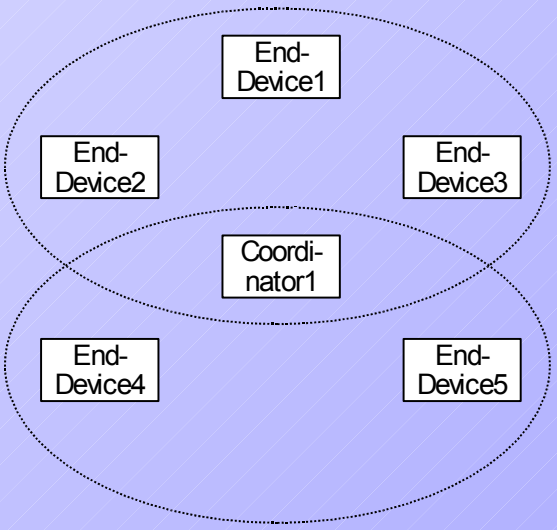
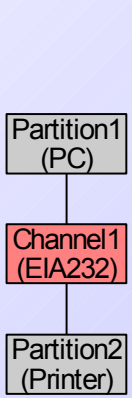
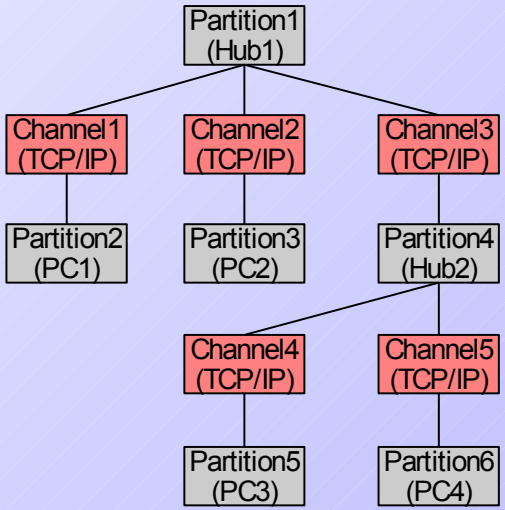
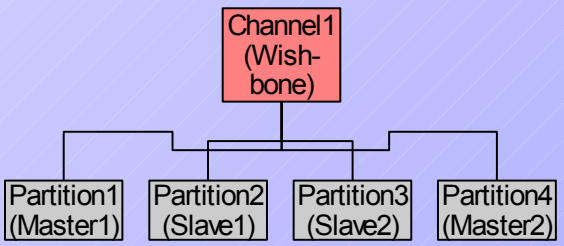
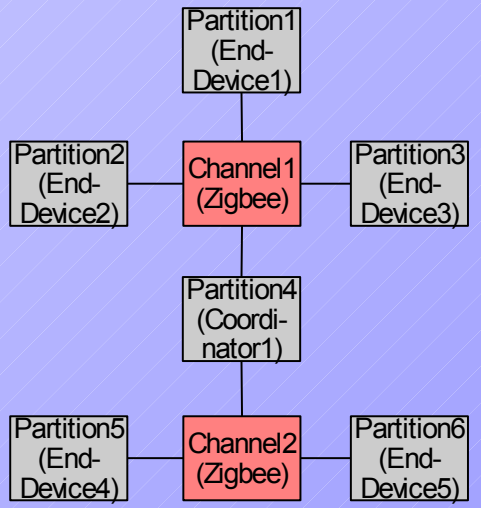
- Partitions as functional resources
  - Modeling of abstract execution components
  - Mapping and execution of functions
  - Determination of execution units for synthesis
- Sender and Receiver
  - Generic partition interfaces to communicate with channels
  - Transformation of functional onto architectural data flows and reverse
  - Usage of channel packets with data, timing, routing and addressing informations
- Channels at individual abstraction level
  - Modeling of communication components (passive, active) to connect partitions (n to n)
  - Internal generic channel interface to communicate with partitions
  - Performance and interconnection schemes
  - Determination of bus types for synthesis



Architecture Model - BM2AMGenerator

- Preparation of the Behaviour Model by User
  - Assignment of partitions to blocks by annotations (e.g.: partition=CPU1;)

- Automated generation of the Architecture Model
  - Replacement of cross-partitional Arguments (Memory, Event, Resource) to enable direct channel communication
  - Create hierarchical blocks for defined partitions
  - Insert depending blocks into partitions
    - assigned functions
    - sender
    - receiver
  - parametrization of sender blocks to assign receiver blocks
  - Connect partitions by default channel blocks

EIA232	Ethernet (or USB20)	Wishbone (SoC)	Zigbee (wireless)
			
			

Architecture Model - Delay Computation

Passive bus (e.g. EIA232)

- ESLChannelSender receives data from a functional source and sends a ESLChannelPacket to ESLChannel
- ESLChannel sends the packet to the ESLQueue
- ESLQueue informs bus specific blocks (observers) that a new packet has arrived
- Some of the bus specific blocks get packets from ESLQueue and compute the performance delay
- One bus specific block sends the delayed packet to ESLChannel
- ESLChannel sends the delayed packet to the correct ESLChannelReceiver

Active bus (e.g. USB20)

- ESLChannelSender receives data from a functional source and sends a ESLChannelPacket to ESLChannel
- ESLChannel sends the packet to the ESLQueue
- Some of the bus specific blocks poll periodical (internal scheduling) ESLQueue to get specific packets and computes the performance delay
- One bus specific block sends the delayed packet to ESLChannel
- ESLChannel sends the delayed packet to the correct ESLChannelReceiver

Step 3: **Performance analysis**

- Processing time and structure of information, degree of attainable parallelism and necessary performance
- Analysis of time-related Architecture Model by simulation
  - Queue size of channels
  - Usage rates and resource utilization
- Refinement of execution and communication architecture
  - Parametrization and replacement of channels (Wishbone, EIA232, USB20, FlexRay, Zigbee, Ethernet)
  - Parametrization of channel queues
  - Modification of resource linking states
- Usage of existing buses (Network Block Set) or definition of new buses (using pattern NBS)

Architecture Model - Simulation

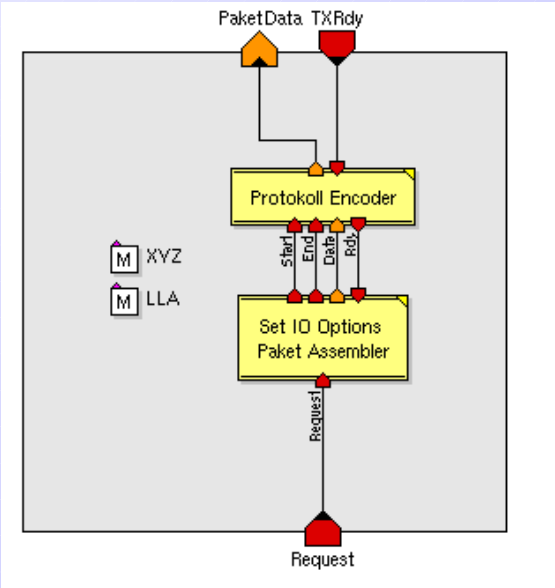
- Before simulation
  - Validation of network by type of connected blocks
  - Address resolution by sender parametrization or parsing of Behavioral Model by XSLT (location of connected ports)
  - Validation of addressing informations
  - Adaption of port data types of sender and receiver functions regarding the connected function to enable simulation
  - Initializing of channel port pointer map to speed up simulation
- During simulation
  - Receive and send channel packets
  - Management of packets by queue
  - Active or passive treatment of packets
  - compute channel specific delays and send delayed packets

Architecture Model - Design of abstract buses

- Determine individual channel abstraction level and find performance related parameters to compute delay
  - Minimum and maximum partition count
  - Active or passive channel behaviour
  - Channel specific resources and coherences
- Compliance to existing channel pattern
  - Each channel needs exactly one block of type ESLChannel
  - Channels can only be connected to partitions with internal blocks of type ESLSender and ESLReceiver
  - Usage of data structure ESLChannelPacket
    - No adaption of partition interfaces necessary
    - Simply usage of channel pattern

Step 4: **Synthesize model into implementation**

- 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

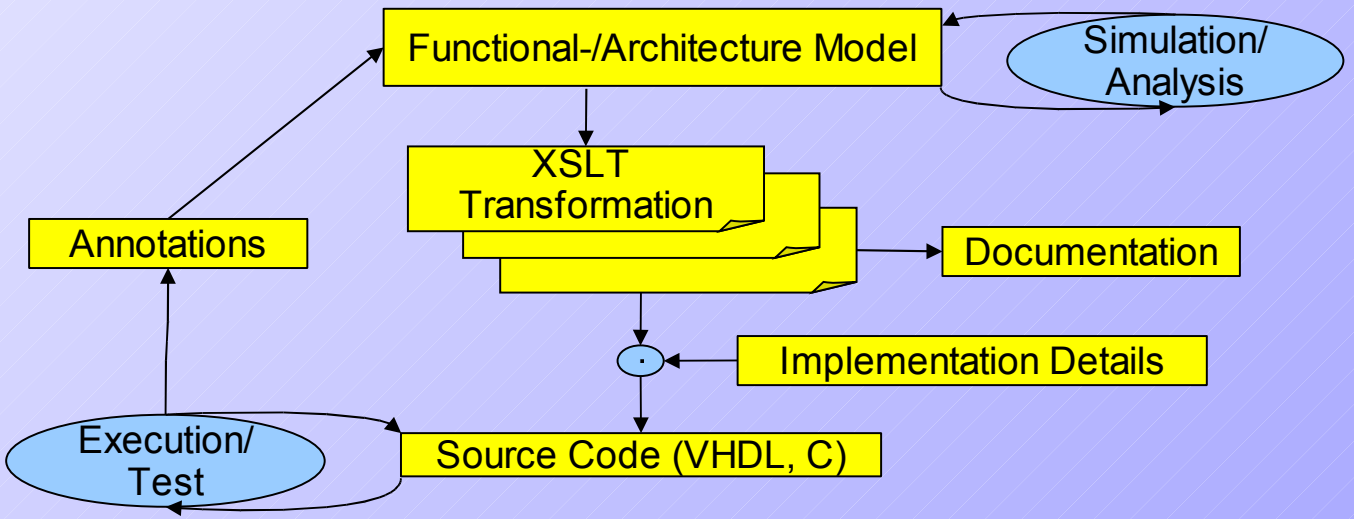


Example

```
-- 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;
```

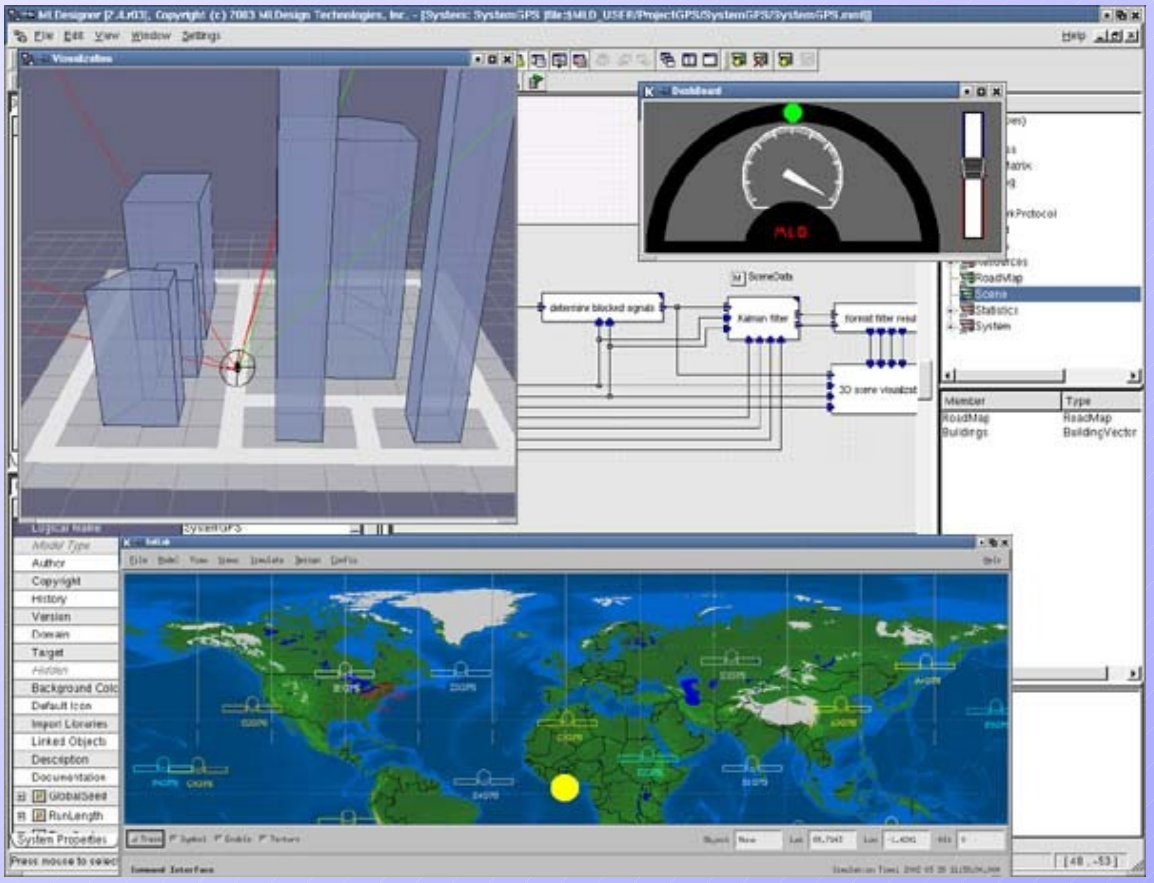
Step 5: **Test and validate prototype**

- 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

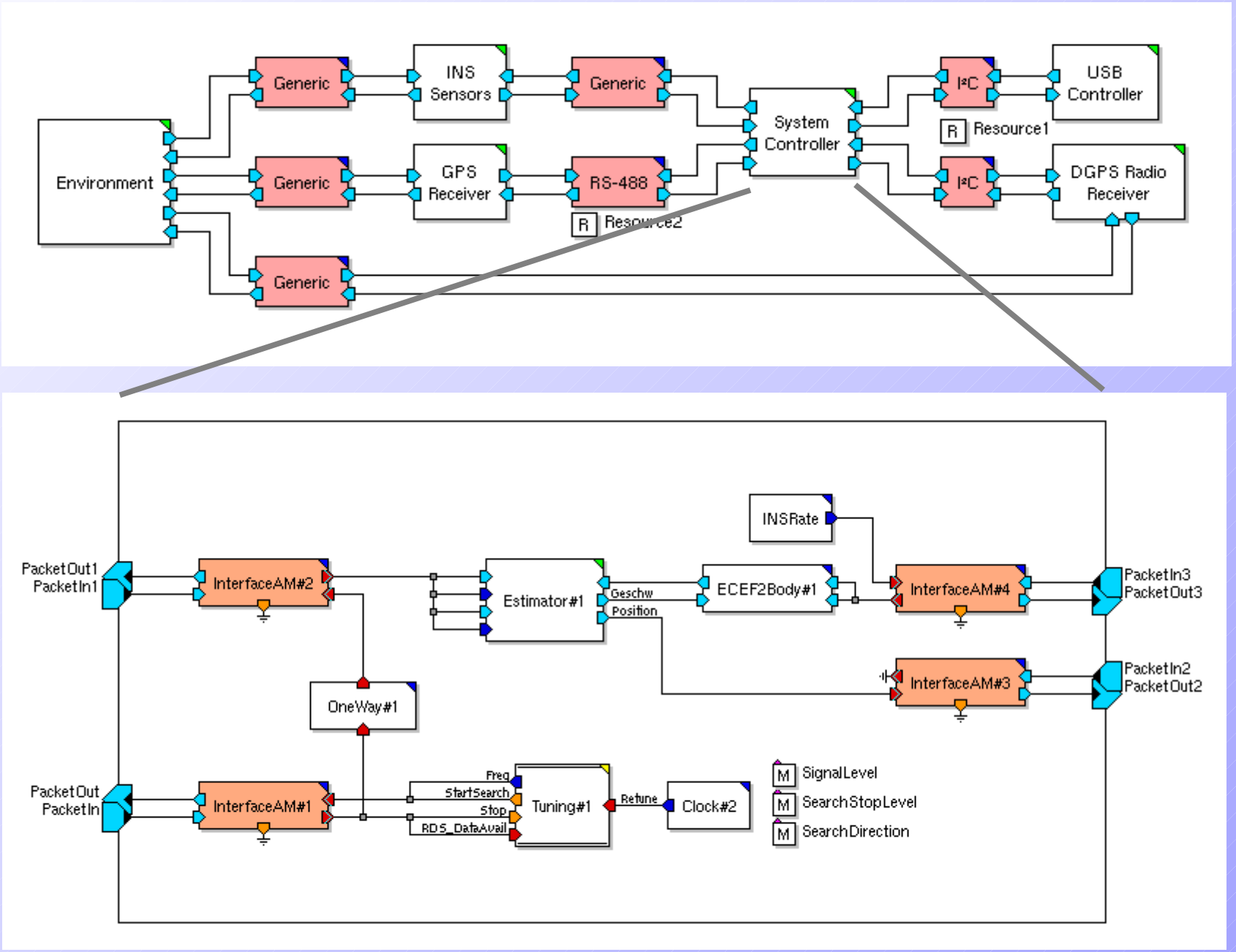


Virtual SatNav Validation Environment

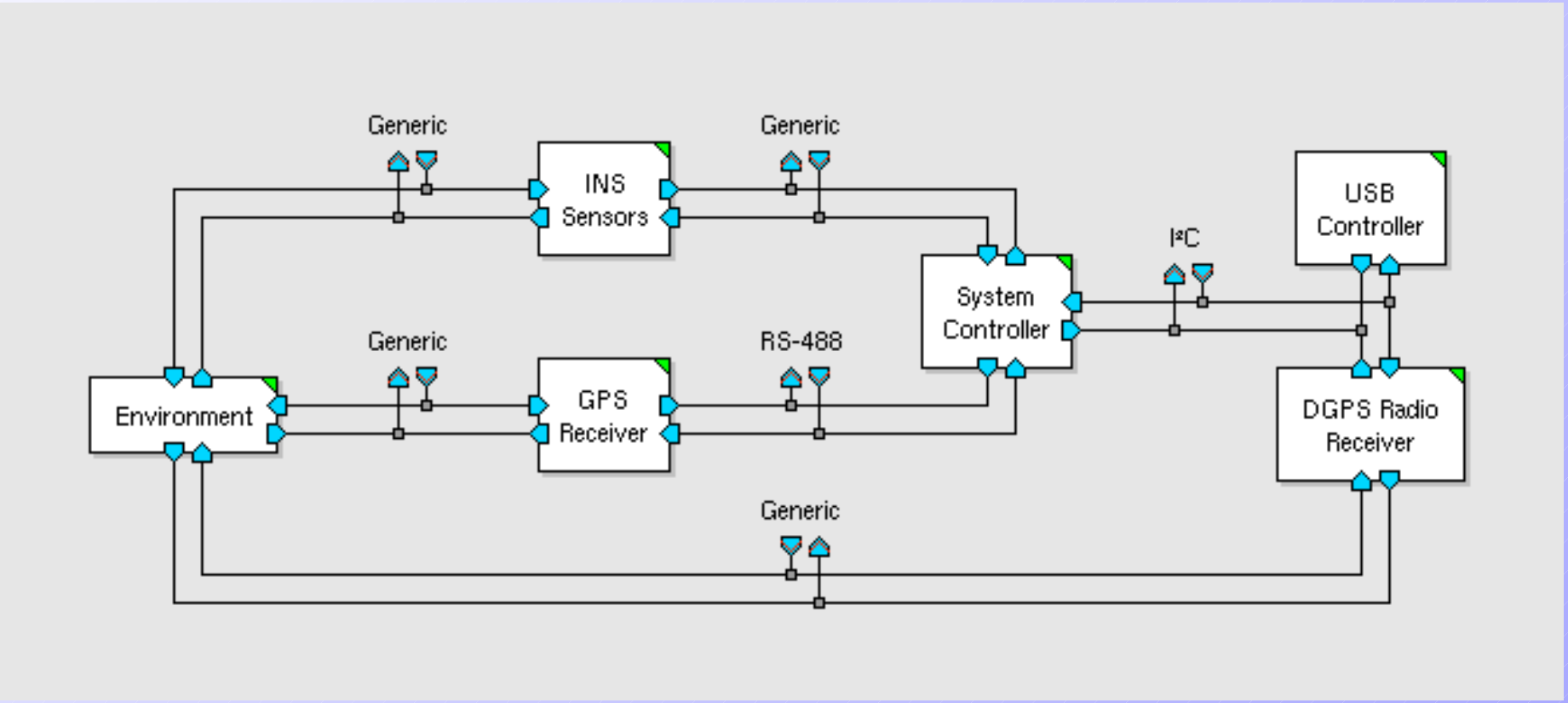
- navigation system model (MLDesigner)
- navigation satellite model (SatLab)
- data driven 3D-scene (OpenGL)
- car controller widget (Tcl/Tk)



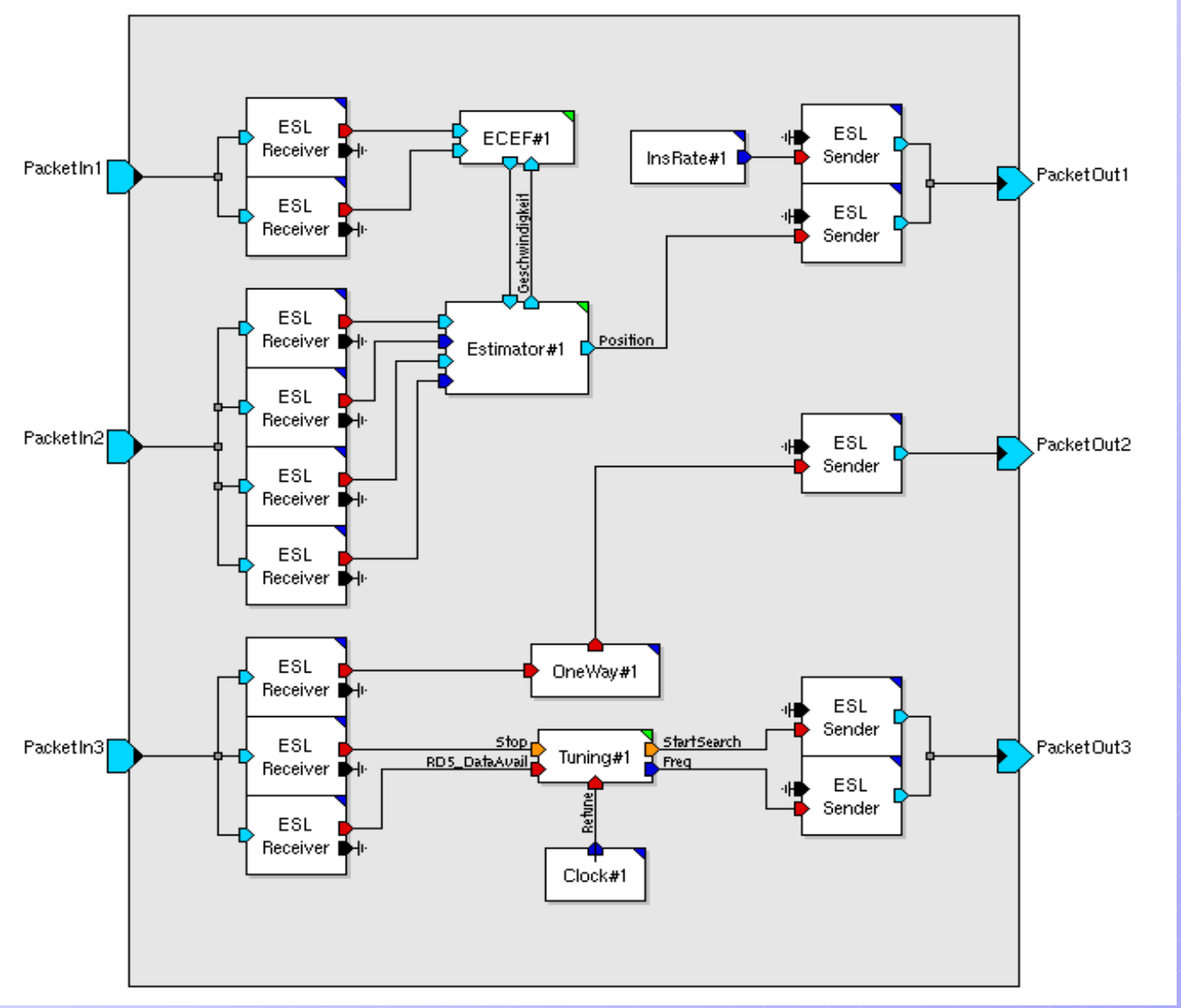
3. Example


Environment
Generic
Generic
Generic
INS Sensors
GPS Receiver
Generic
Generic
RS-488
R Resource2
System Controller
I²C
I²C
R Resource1
USB Controller
DGPS Radio Receiver
PacketOut1
PacketIn1
InterfaceAM#2
Estimator#1
Geschw
Position
ECEF2Body#1
INSRate
InterfaceAM#4
PacketIn3
PacketOut3
InterfaceAM#3
PacketIn2
PacketOut2
OneWay#1
PacketOut
PacketIn
InterfaceAM#1
Freq
StartSearch
Stop
RDS_DataAvail
Tuning#1
Retune
Clock#2
M SignalLevel
M SearchStopLevel
M SearchDirection

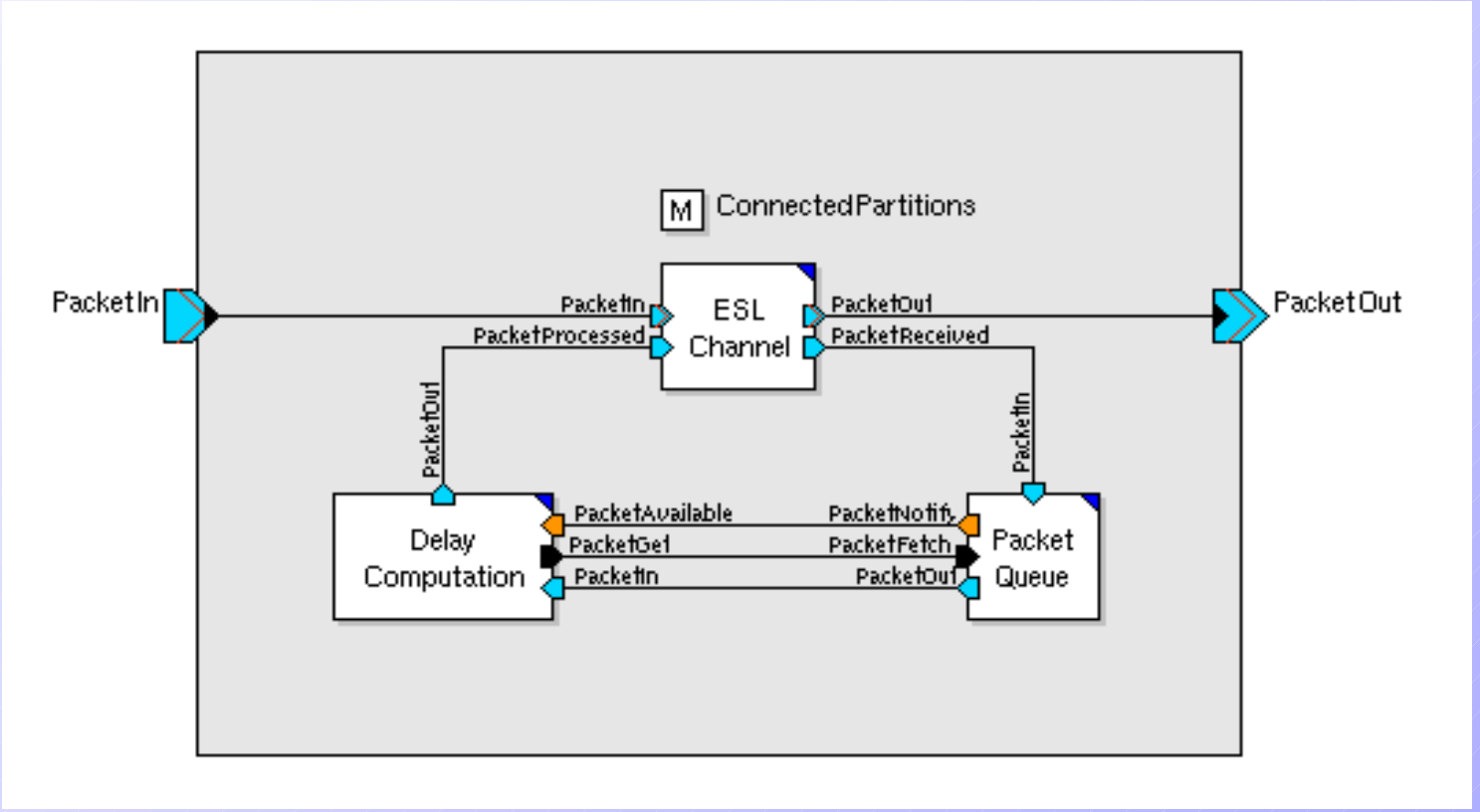
Top-Level Chippo Model

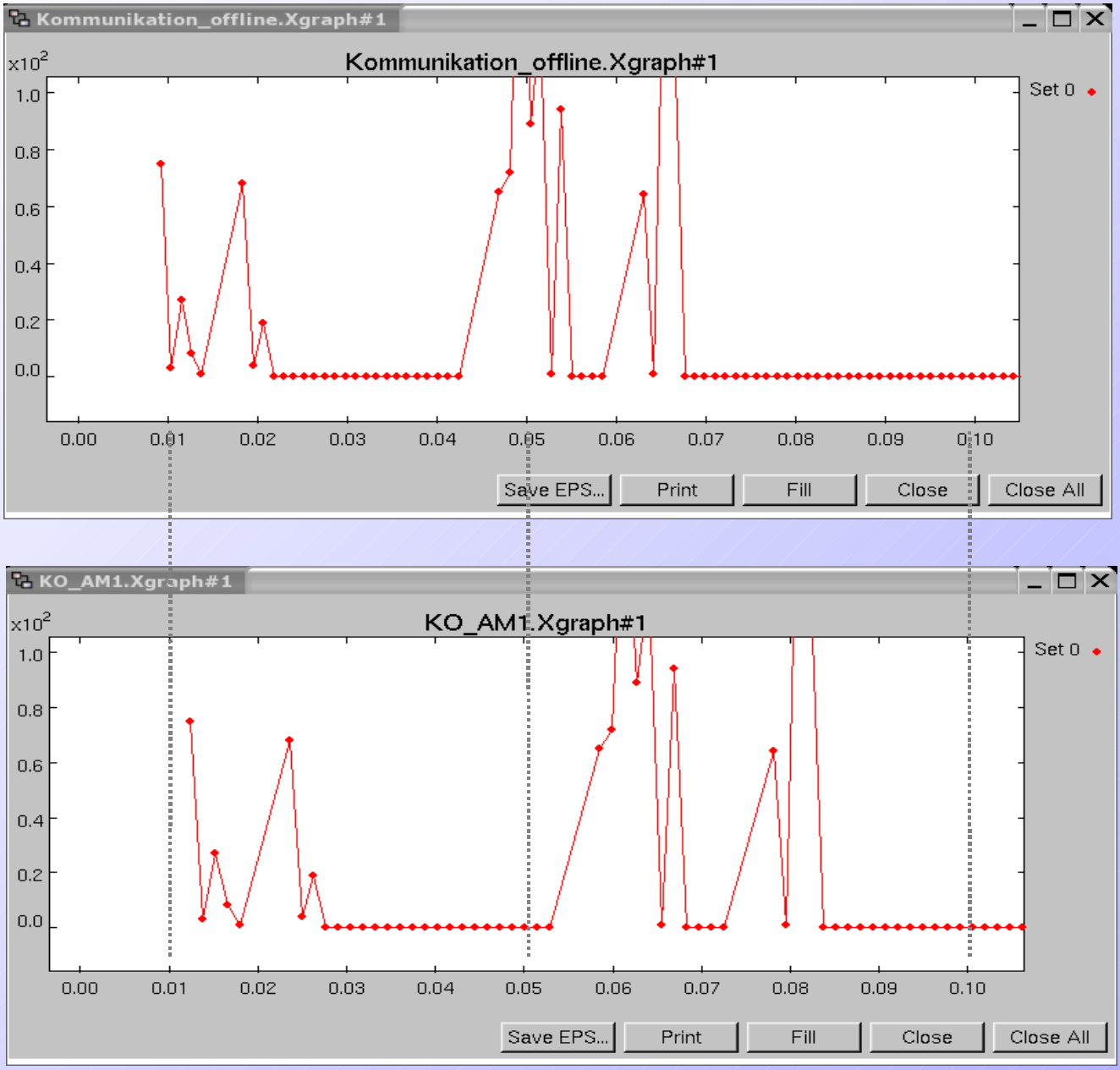


Partition System Controller



I²C Channel


M
ConnectedPartitions
PacketIn
PacketIn
PacketProcessed
ESL
Channel
PacketOut
PacketReceived
PacketOut
PacketIn
PacketIn
Delay
Computation
PacketAvailable
PacketGet
PacketIn
PacketNotify
PacketFetch
PacketOut
Packet
Queue

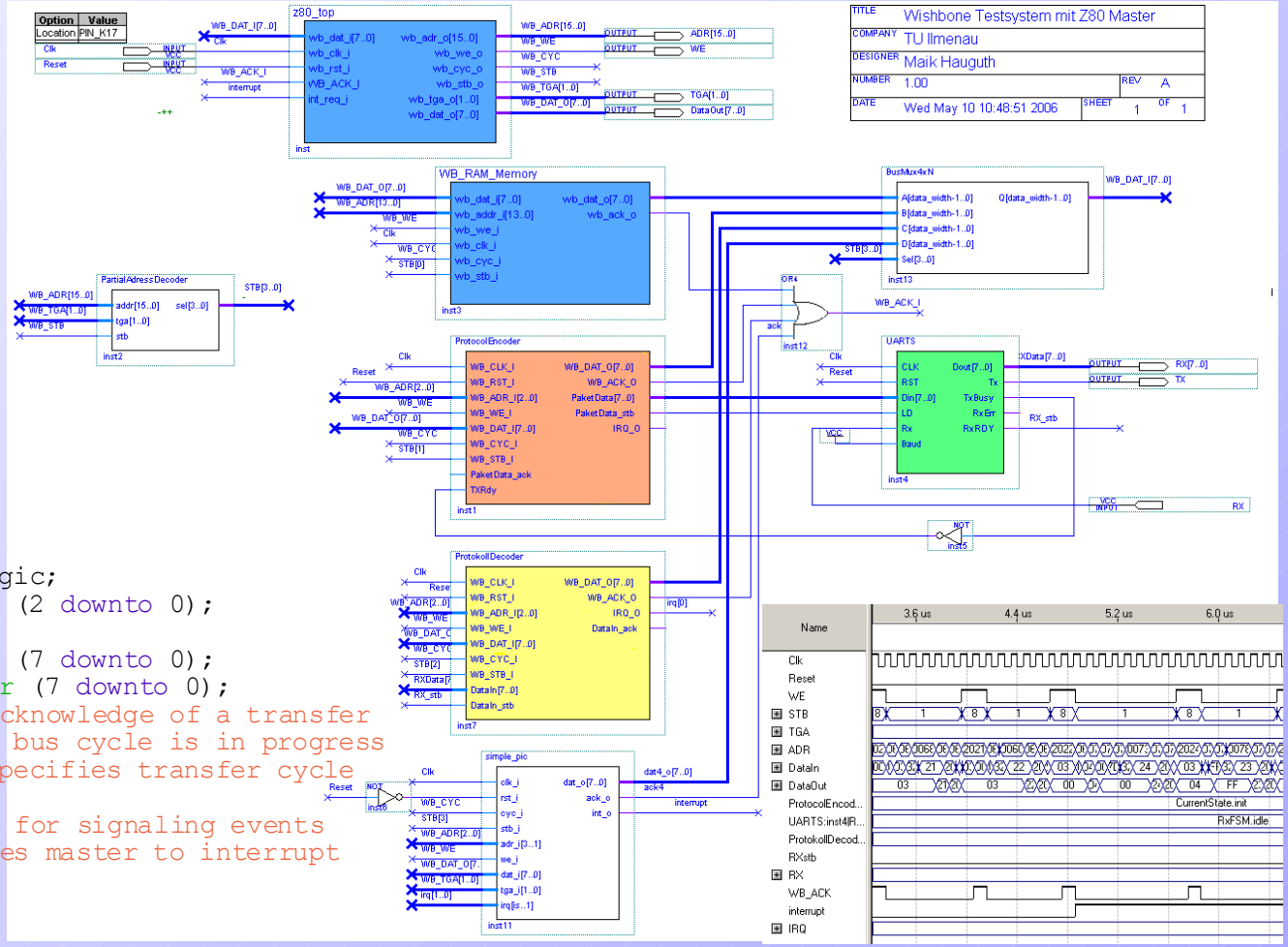


Without Architecture Model

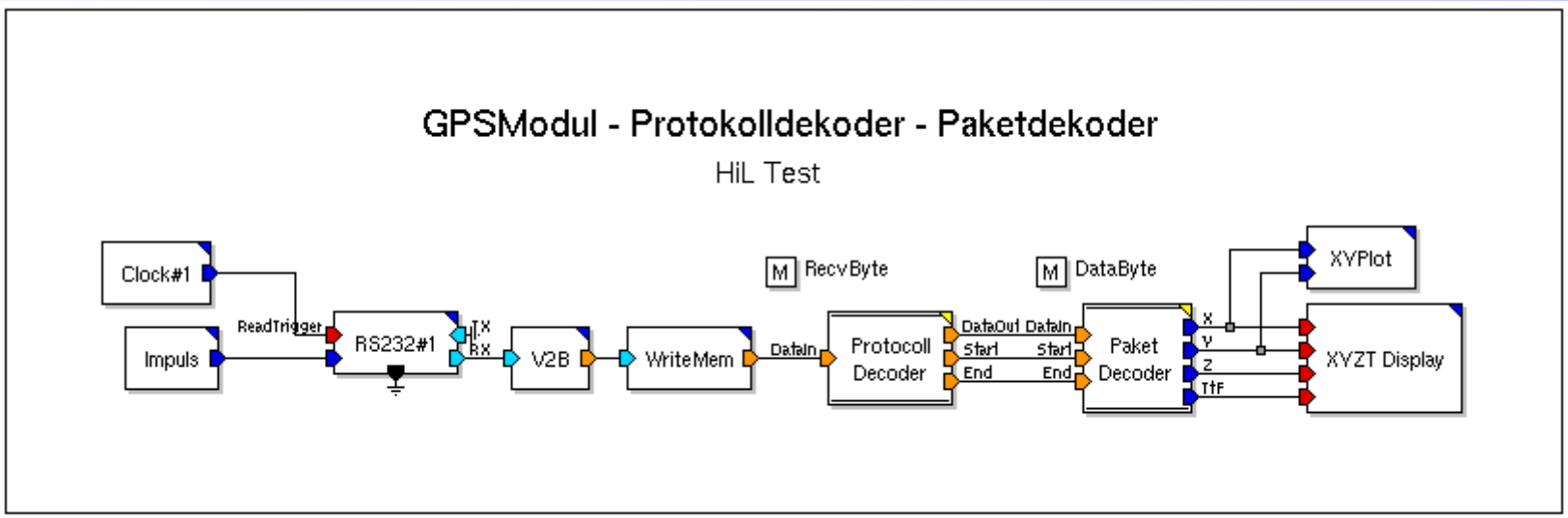
With Architecture Model
(channel specific delays)

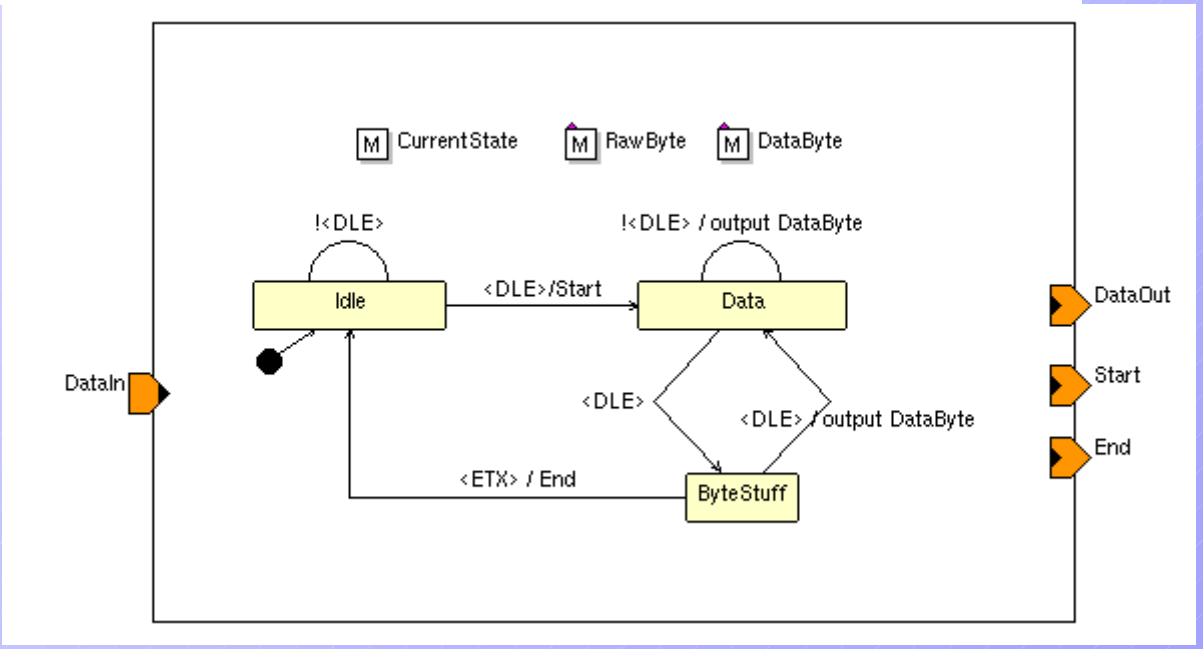
FPGA Implementation

```
-- 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 simulation





Summary

- A methodology was presented to design at ESL and map the design into an implementation using MLDesigner:
  1. functional design with virtual GPS development system
  2. automatically mapping of plug and play HW models into functional design
  3. iterating on integrated HW/SW model until functional requirements are met with the selected HW
  4. semi-automatically mapping selected partitions into VHDL -> FPGA
  5. hardware in the loop simulation with FPGA
- The methodology was validated for an example of a GPS based navigation system with INS and differential signal correction