Ultra Electronics Holdings plc

The Ultra approach to Model Based Design for safety-critical FPGAs

MATLAB Expo 2018

Process Justin Lennox

FPGA David Amor

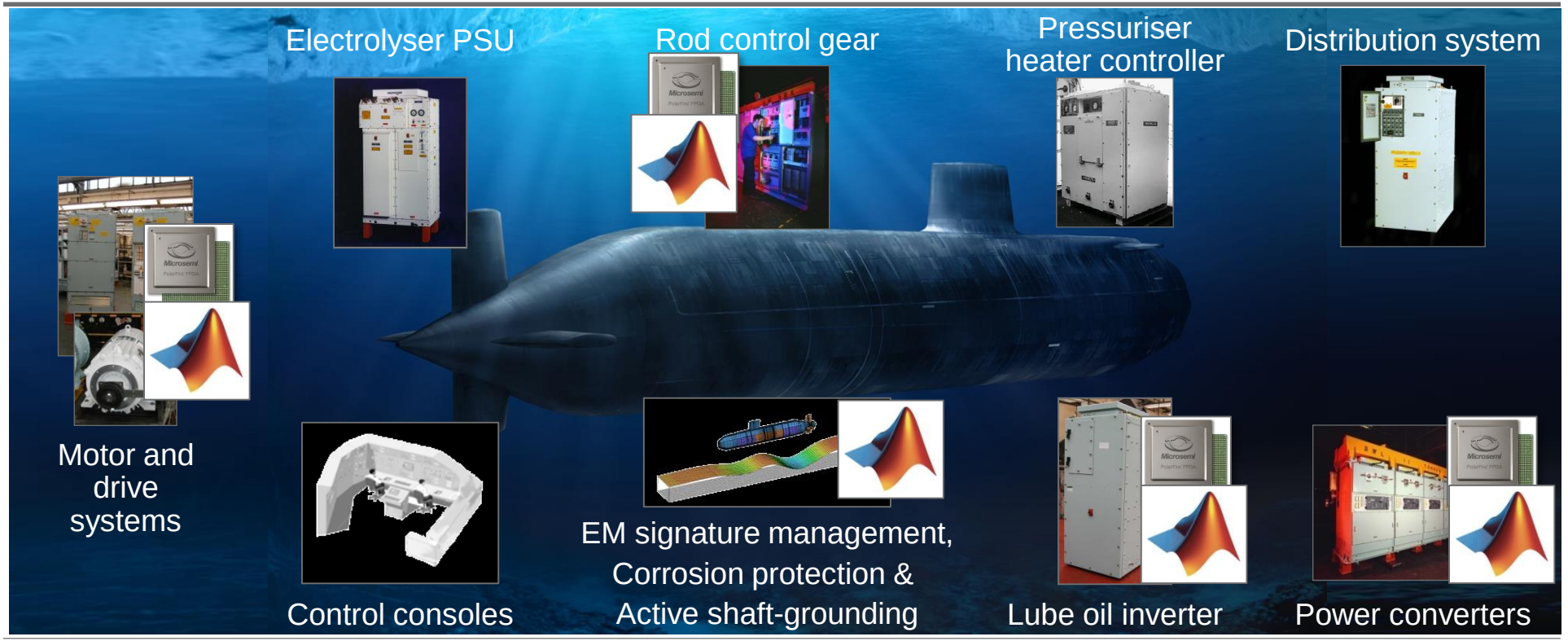
Ultra Electronics Holdings plc

About Ultra Electronics PMES

Justin Lennox

PMES scope of supply

Submarine example

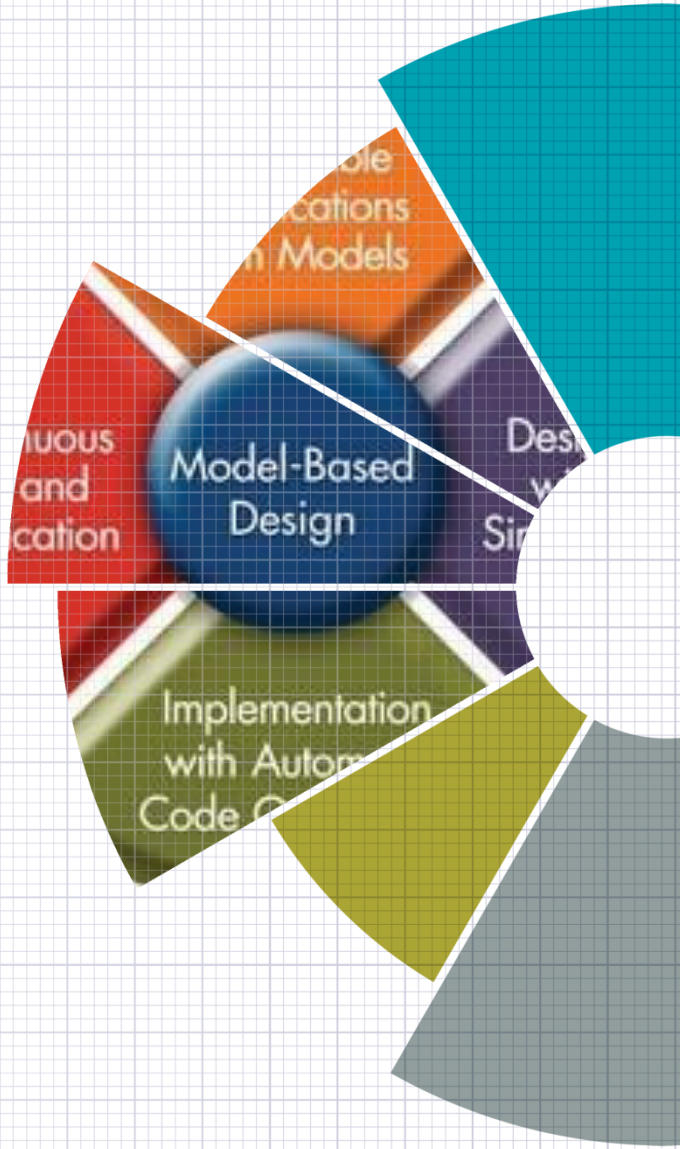


Ultra Electronics Holdings plc

The Ultra approach to Model-Based Design

Applying the MBD process

Justin Lennox



Model Based Design

What and why

- From MathWorks®:

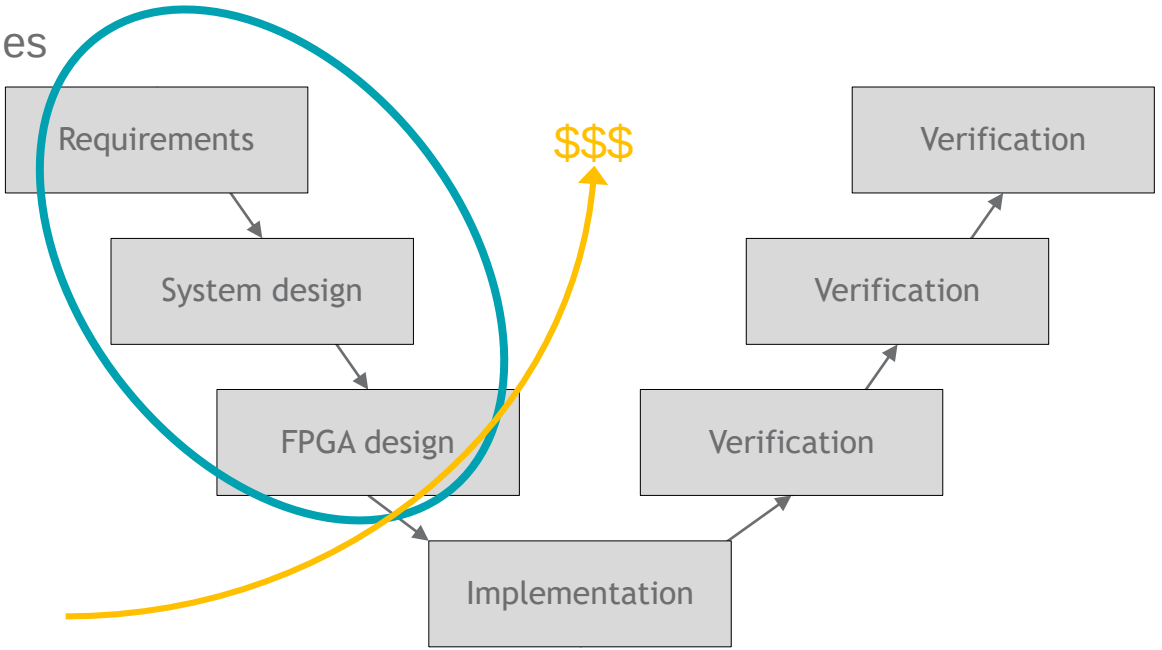
  "*In Model-Based Design, a system model is at the center of the development process, from requirements development, through design, implementation, and testing.*"

- Helps us deal with complexity
- Can test requirements early
- Makes dealing with change easier
- Get things [more] right first time

System design process

Today's focus

- Use of MBD
  - From requirements to realisable modules
  - Increasing cost of bugs
- Supporting functions
  - Design assurance for high integrity systems
  - Long term support



Traditional design process

Pros and cons

Pros	Cons
Everything is written down	Misinterpretation of requirements possible
Documentary evidence easily available	Easy to overlook gaps, contradictions or emergent behaviours in requirements
No expensive tools needed (documents and spreadsheets)	Bugs may only be identified during hardware testing (exponential cost)

Model Based Design

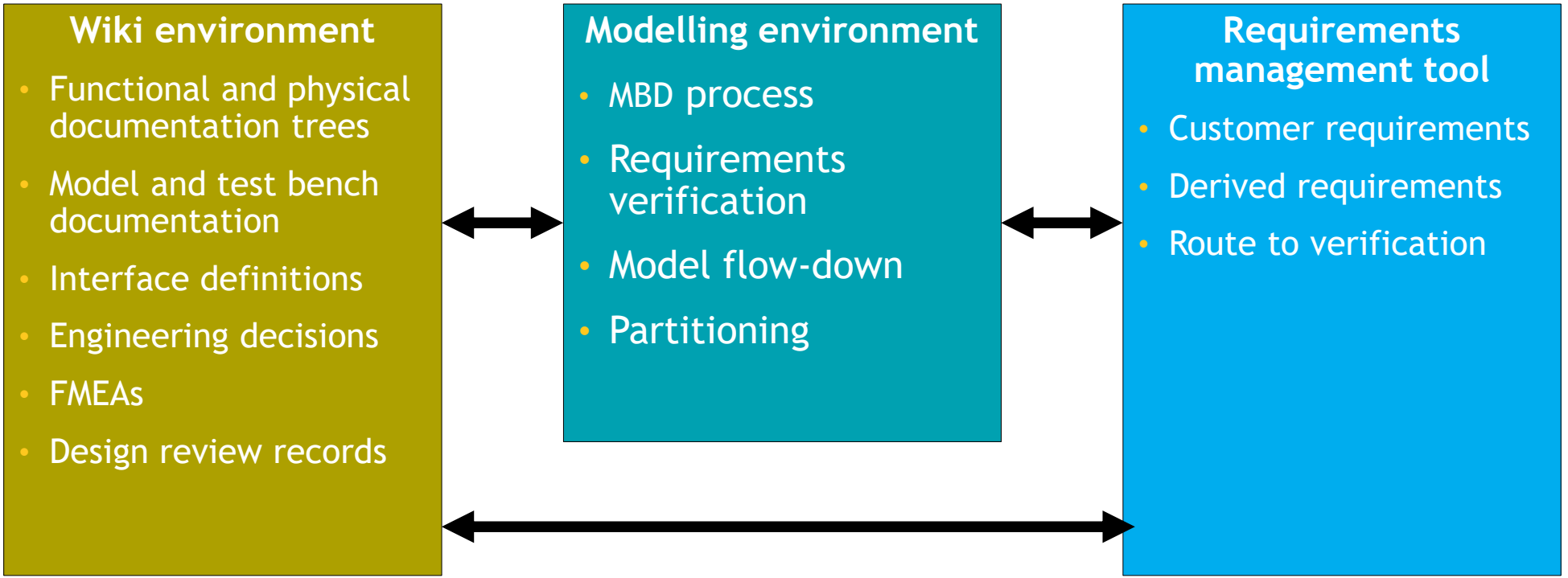
Is it the whole answer?

- No interpretation of requirements (Executable models describe the requirements and design)
- Requirements can be tested/verified throughout – problems found early

But…

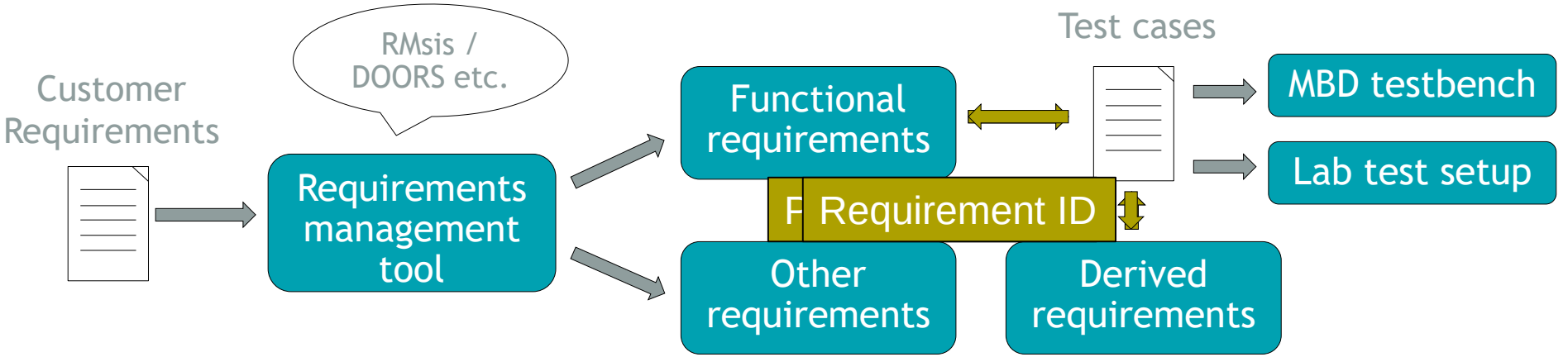
- Need documentation for design assurance
  - Evidence that the system is well defined
  - Evidence for rigorous process
- Need documentation for long term support
  - Design decisions, rationale
- Need to make information available to everyone on the team
  - (Not just those with modelling environment)

Supporting activities



Requirements management

Test cases tie everything together

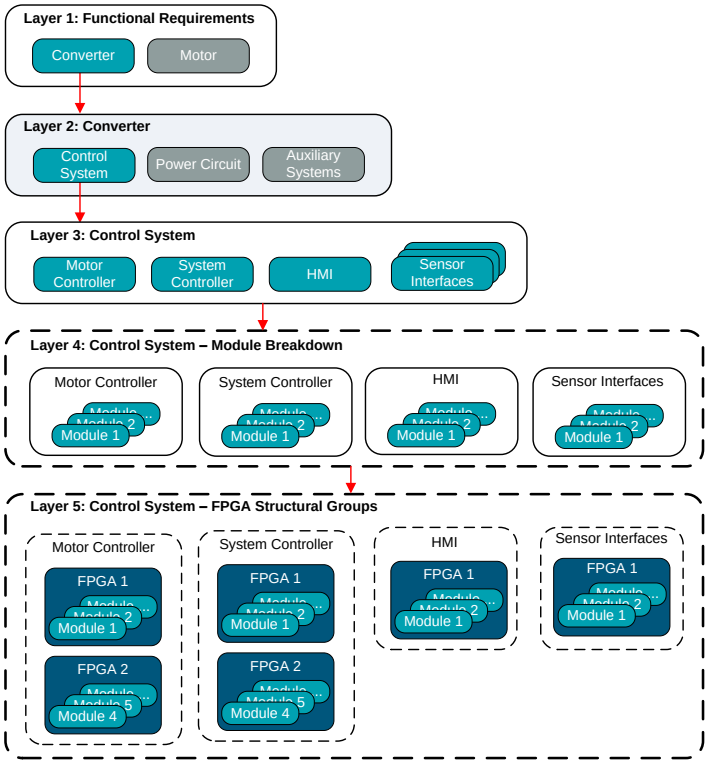


- Functional customer requirements have test cases assigned
- Pass fail criteria for test cases are provided by customer or derived requirements
- Where possible, simulation test cases should match lab tests

Worked Example

Motor converter

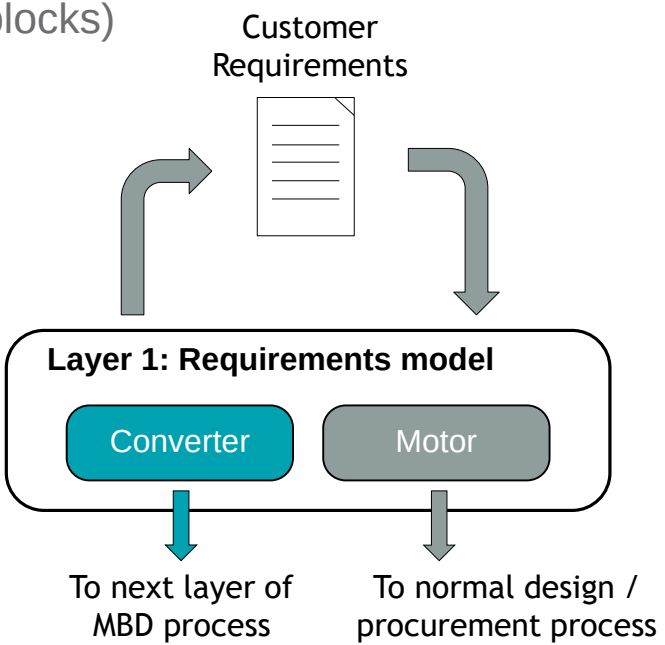
- Requirements captured as a model
- Broken down into functional blocks and modules
- Need some idea of how the equipment will be physically built
- Verification takes place at each layer



Layer 1 – Requirements model

Requirements capture and feedback

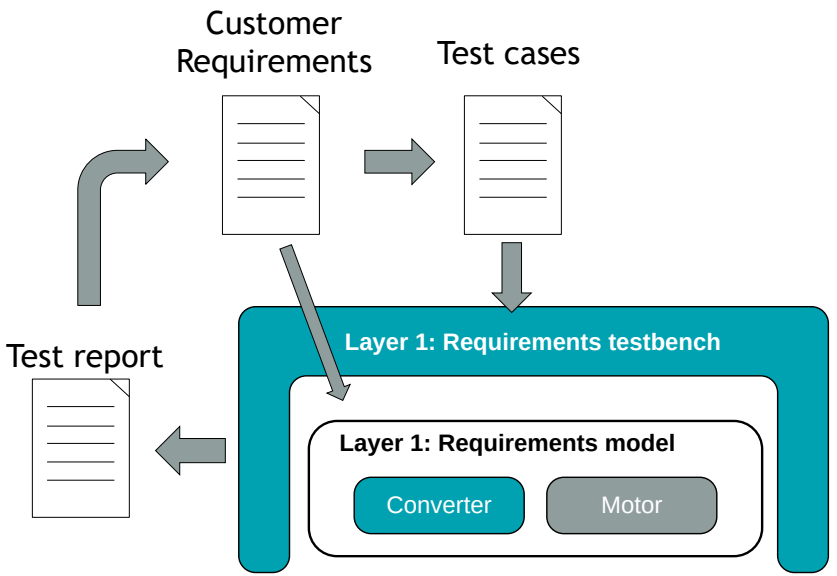
- Functional requirements turned into a Simulink® model
  - Floating point, variable step size
  - Use most convenient tools (Simulink, Stateflow, MATLAB code blocks)
  - Use referenced model to allow use in different testbenches
- Important to feed back at this stage! Iterate to remove:
  - Contradictory requirements
  - Undefined area of operation
  - Unforeseen behaviours
- Design decisions and assumptions recorded and brought off by stakeholders as required
- Move equipment with controllers to the next layer



Layer 1 – Requirements model

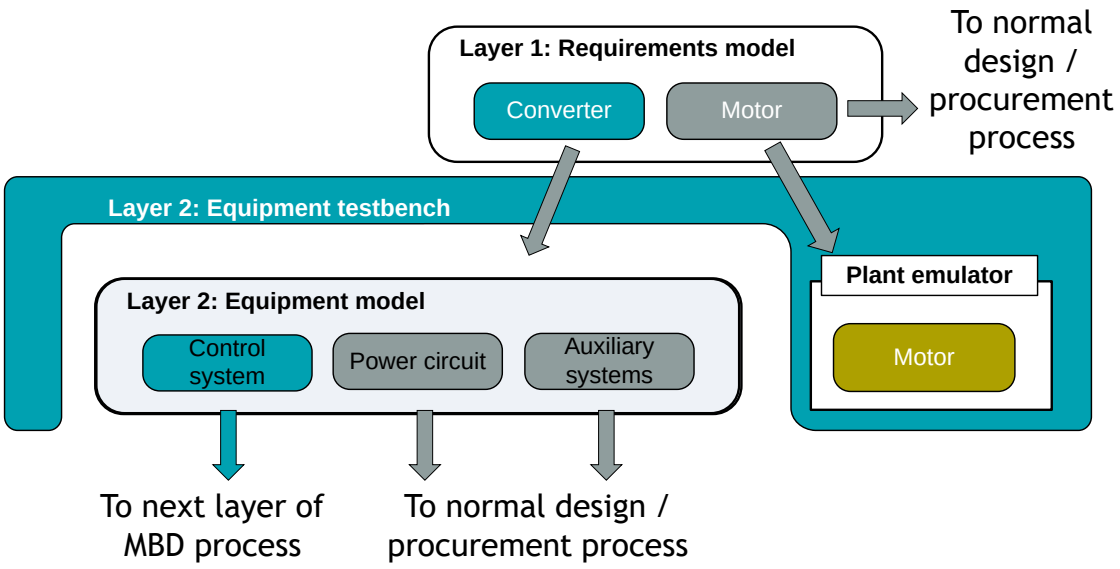
Testbench

- Testbench built from requirements by independent engineer
- Tests only affect the external interfaces
- Good idea to automate testbench
  - Allows easy regression testing
  - Automated report generation
- Source control - critical to have confidence & transparency in generated results



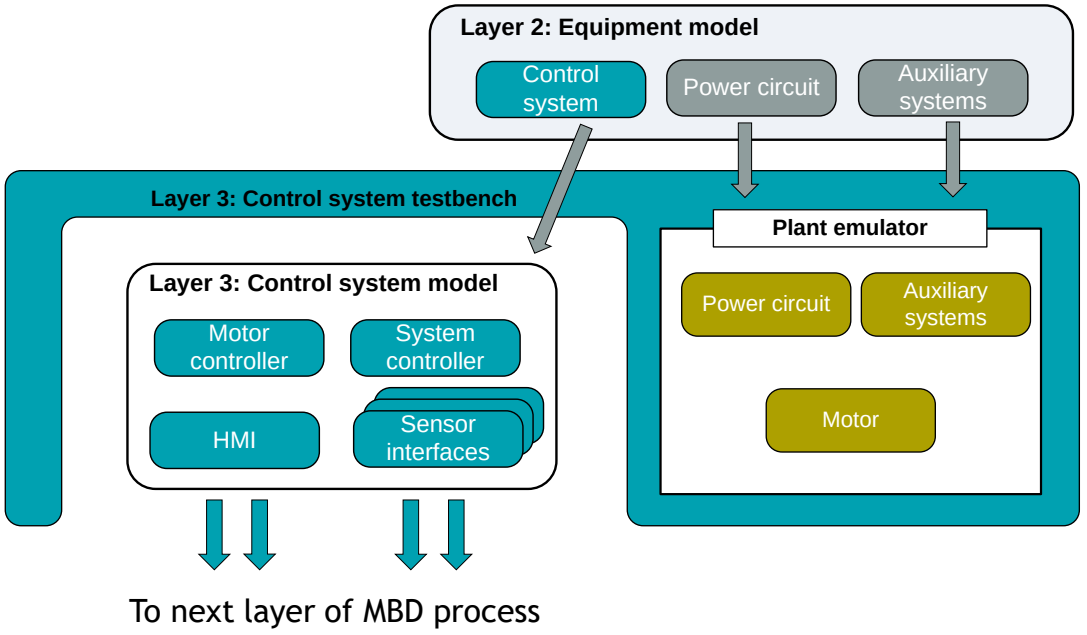
Layer 2 – Equipment model

- Requirements model broken up into individual equipment
- Interfaces between equipment in the system defined at this stage
  - Trivial in this example but can be complex when multiple equipment with controllers exist in the system!
- Testing can now exercise interfaces between equipment
- Move control system(s) to next layer



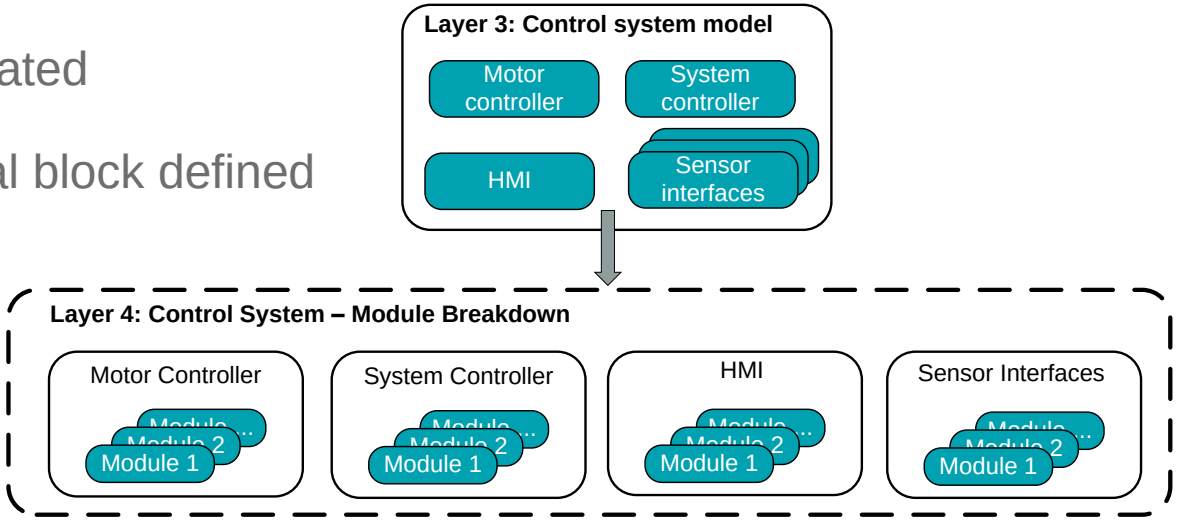
Layer 3 - Control system model

- Control system broken out from other subsystems
- Control system interfaces defined and tested at this stage
- The control system is tested and verified



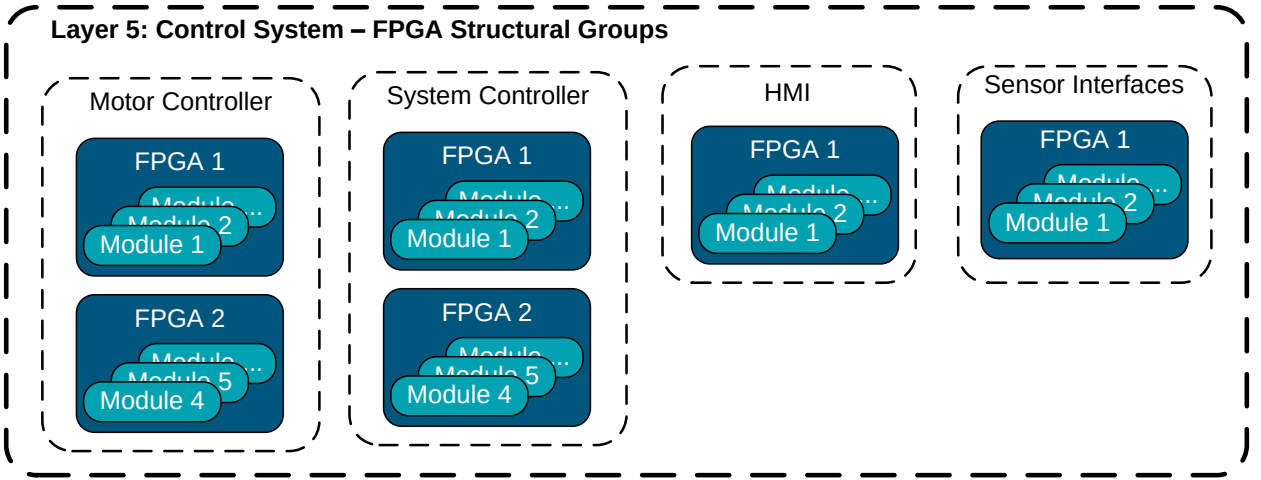
Layer 4 – Functional block models

- Set of functional block models created
- Interfaces between each functional block defined
- Functional blocks tested



Layer 5 – Modules assigned to FPGAs

- Modules that make up each functional block assigned to FPGAs
- Model converted to use fixed point maths (if not already done)
- Interfaces between each module defined
- Bit interfacing for fixed point model.
- At some point need to get to fixed step.



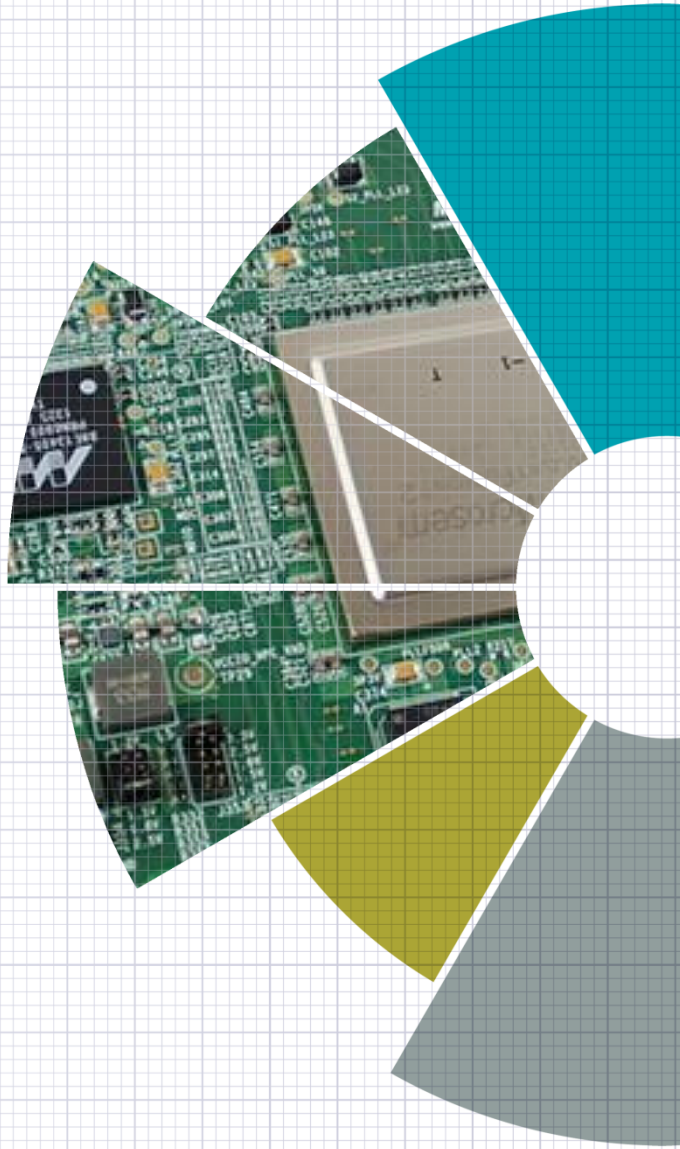
Ultra Electronics Holdings plc

FPGA

Development in a MathWorks Environment

With alignment to IEC61508

David Amor



10 years with MathWorks

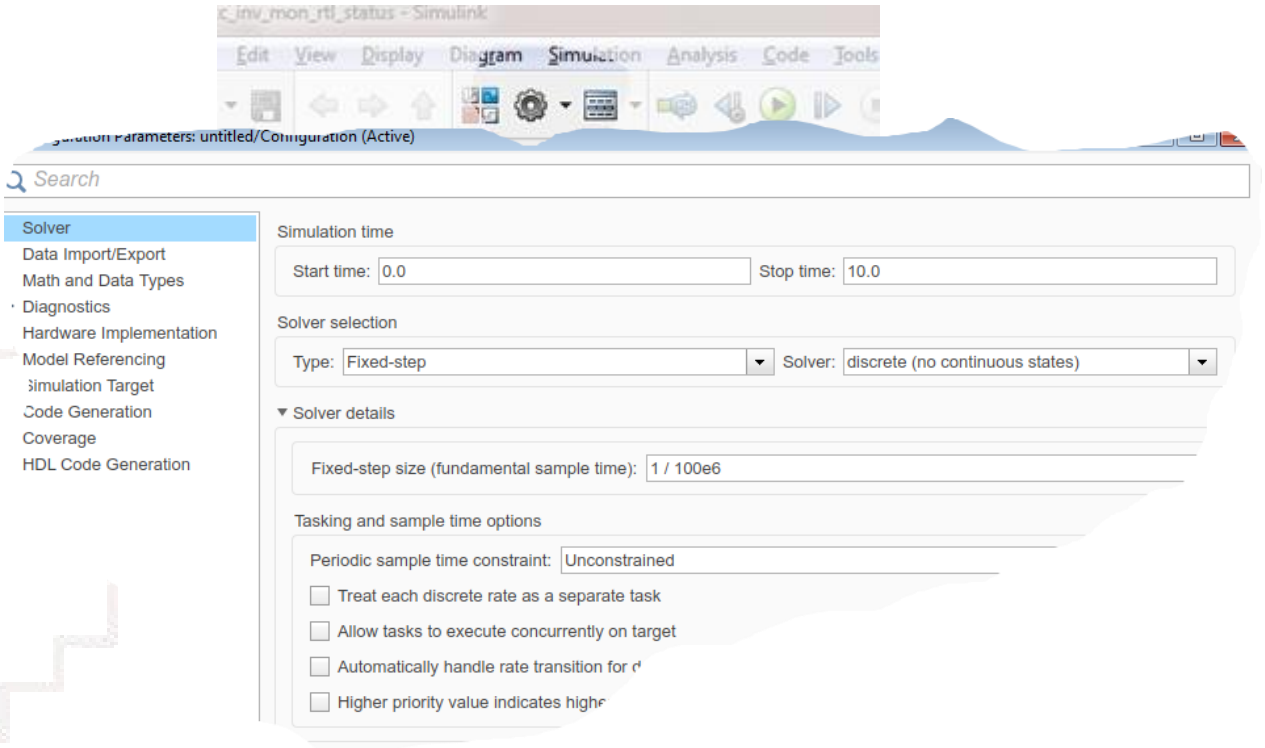
The mechanics of FPGA production

1. Tool: Simulink fixed solver discrete step
2. Design: Schematics for Architecture
3. Design: **E**mbedded **M**AT**L**AB (EML)
4. Design: Stateflow
5. Design: Using Buses
6. Design: Reuse - Model References, libraries
7. Tool: Scripting (**H**ardware **D**escription **L**anguage) HDL generation
8. Tool: Projects and change control with (**S**ub**v**ersio**n**) SVN with Jira
9. V&V: Test benching and **Model** coverage
10. V&V: Co-simulation of generated HDL and **Code** coverage

Simulink fixed solver discrete step

Fixed clock period for synchronous designs

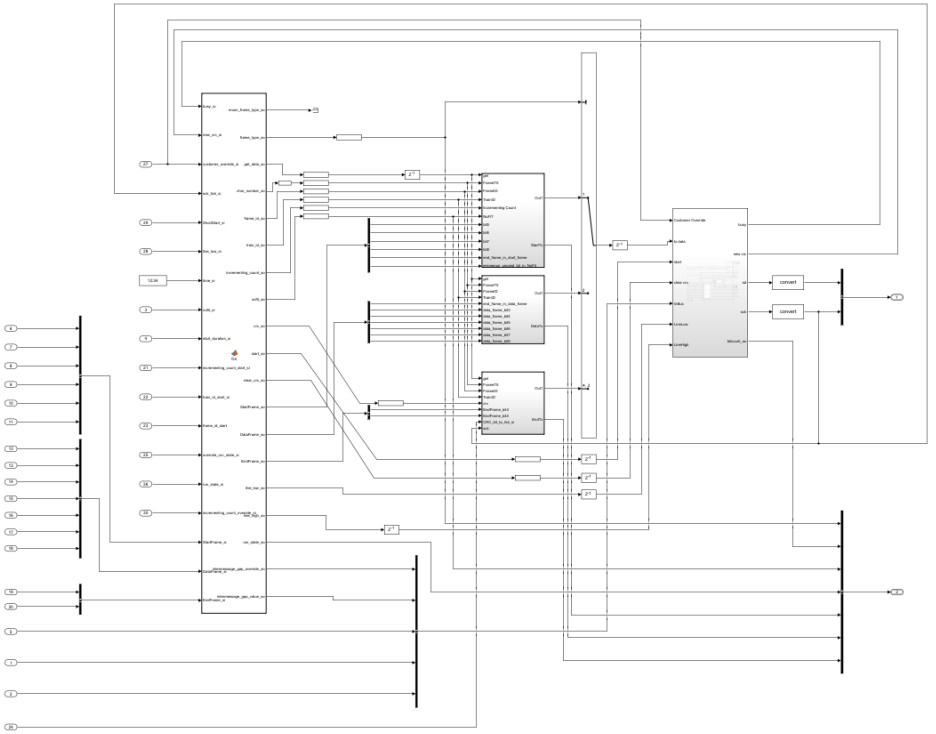
Ensure consistent
Results when co-simulating



Schematics for Architecture

Architecture describes the signal flow between functional blocks

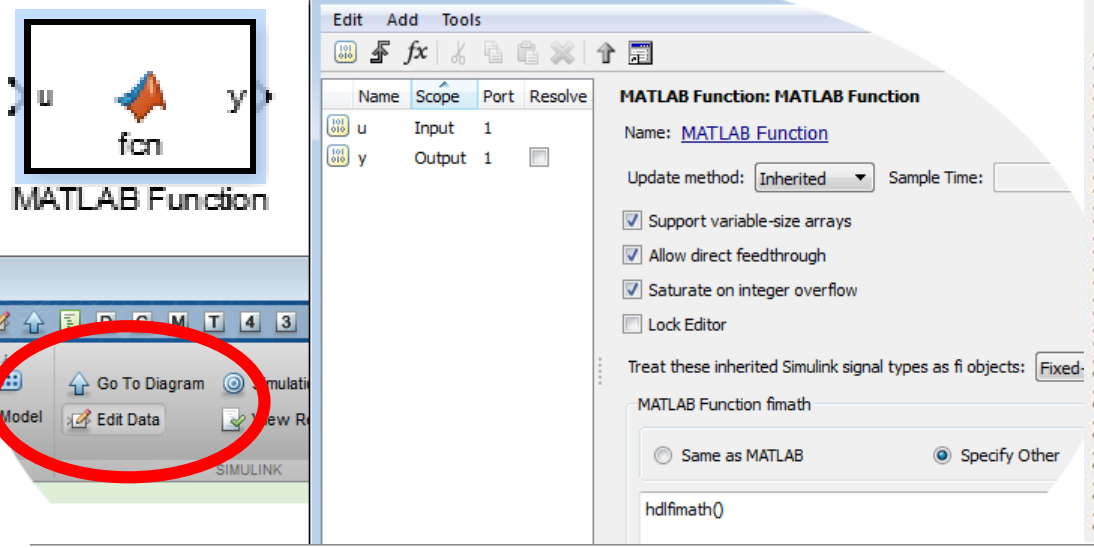
- Mixture of:
- EML
- Subsystems
- Model references
- Libraries



Embedded MATLAB (EML)

Persistence, if isempty(foo), (:), fixdt

Fixed point data types, Controlling data type bits
The reshape “(:)” operator, Code of practice naming.




```
1   function cntr_so = counter_eml(end_value_si)
2   %#codegen
3    % Counter An 8bit counter that terminates
4    % at "end_value_si" - reseting to 0
5
6    % Declare signals
7    persistent cntr_s
8
9    % Reset
10   if isempty(cntr_s)
11       cntr_s = fi(0,0,8,0);
12   end
13
14   % Define outputs
15   cntr_so = fi(0,0,8,0);
16
17   % Assign outputs
18   cntr_so(:) = cntr_s;
19
20   % The actual function
21   if cntr_s >= end_value_si
22       cntr_s(:) = cntr_s + 1;
23   else
24       cntr_s(:) = 0;
25   end
```

Stateflow

Code diversity for IEC61508 – removing common mode failures

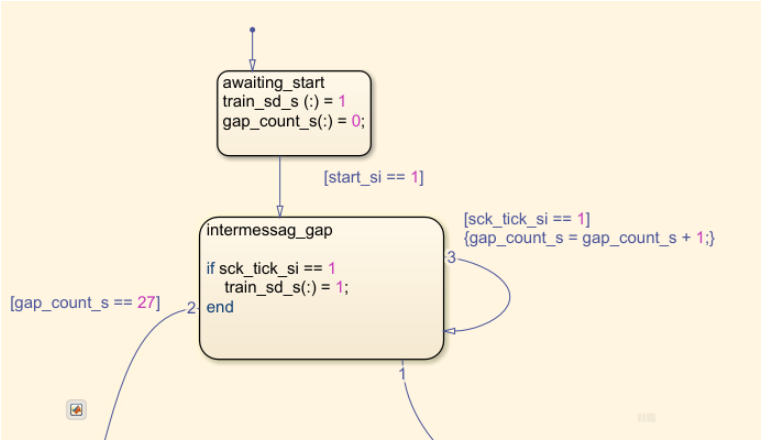
Enforced state machine heterogeneity with equivalent functionality.

e.g. Control / Protection relationship: defensive and diverse code.

```
49 - switch state_s
50
51     case 0 % awaiting start (incl intermessage gap)
52
53 -       if start_si == 1
54             % (Skip intermessage gap)
55 -           state_s(:) = state_s + 1;
56         end
57
58     case 1 % intermessage gap
59
60 -       if sck_tick_si == 1 % serial interface clock
61 -           train_sd_s(:) = 1;
62         end
63
64 -       if inter_msg_cnt_s > 0
65 -           busy_s(:) = 1;
66
67 -           if sck_tick_si == 1
68 -               inter_msg_cnt_s(:) = inter_msg_cnt_s - 1;
69             end
70
71 -       elseif  inter_msg_cnt_s == 0
```

State flow is synthesizable with caveats

State flow is easier to visualise at simulation time



Using Buses

Tidy schematics and ease of signal maintenance

mat file for bus definition when traversing reference model

Reading / writing to bus from EML:

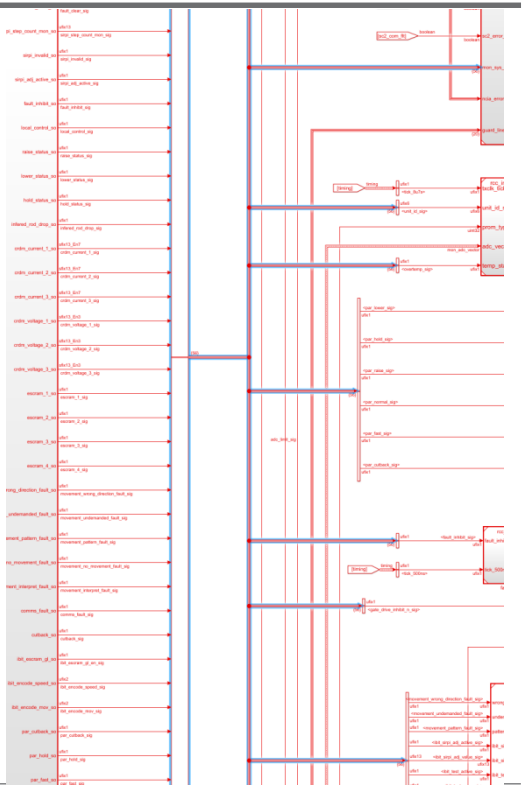
Help checker by constraining the port to use the bus definition in the ports and data manager:

		Output	11	Inherit: Same as Simulink
	StartFrame_so	Output	12	Bus: StartFrame
	DataFrame_so	Output	13	Bus: DataFrame

EML uses dot notation to drill into busses

e.g. Writing: StartFrame.Char2 = fi(85,0,8,0);

Reading: crc_s = EndFrame.Char7;



Model References, libraries

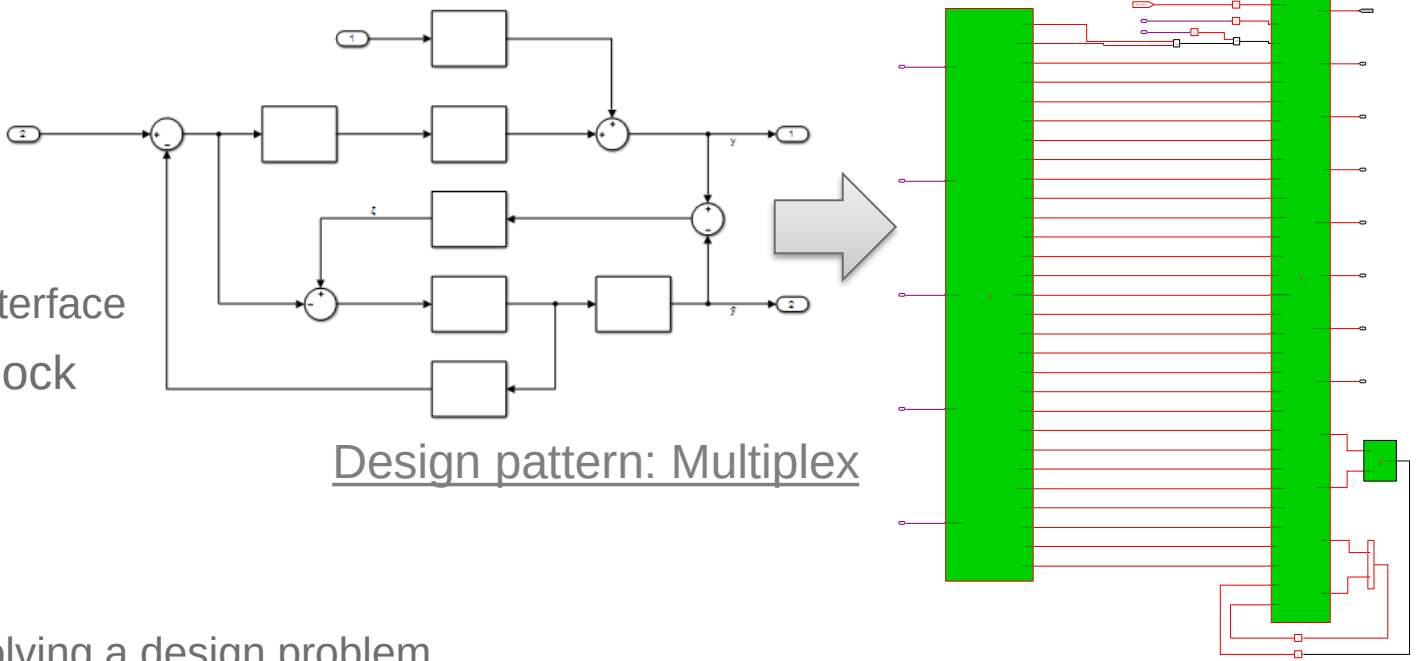
Reusability and module level testing

Division of architecture allowing reuse and easier testing

Four types of ‘code’

1. None repeating
2. Library
   - Project specific
   - e.g. proprietary serial interface
3. Common functional block
   - standalone function:
   - multiplier, metafilter etc
4. Design patterns

Common approach to solving a design problem

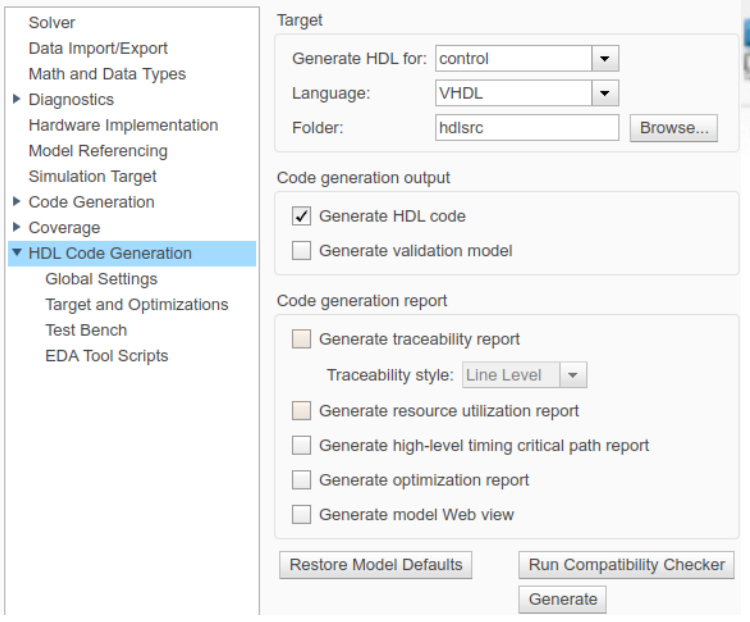


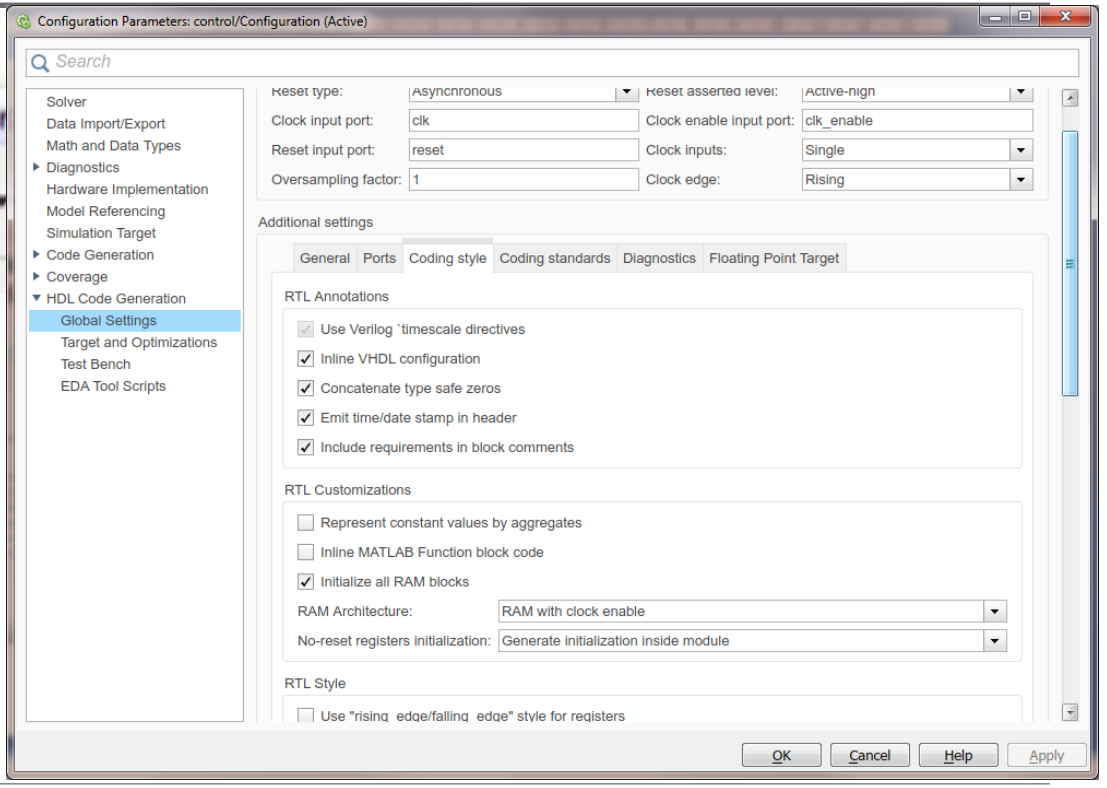
Design pattern: Multiplex

Manual HDL generation

VHDL Output

Model Configuration parameters (cog)
“HDL Code Generation” -> Generate





Resulting VHDL

Comparison of Matlab code with VHDL output

```
LIBRARY IEEE;
USE IEEE.std_logic_1164.ALL;
USE IEEE.numeric_std.ALL;

ENTITY counter_eml IS
  PORT( clk_si           :   IN    std_logic;
        resetn_si        :   IN    std_logic;
        end_value_si     :   IN    std_logic_vector(7 DOWNTO 0);  -- uint8
        cntr_so          :   OUT   std_logic_vector(7 DOWNTO 0)  -- uint8
        );
END counter_eml;


ARCHITECTURE rtl OF counter_eml IS

  -- Signals
  SIGNAL end_value_si_unsigned   : unsigned(7 DOWNTO 0);  -- uint8
  SIGNAL cntr_so_tmp             : unsigned(7 DOWNTO 0);  -- uint8
  SIGNAL cntr_s                  : unsigned(7 DOWNTO 0);  -- ufix8
  SIGNAL cntr_s_next             : unsigned(7 DOWNTO 0);  -- ufix8
```

Projects and change control with (Subversion) SVN with Jira

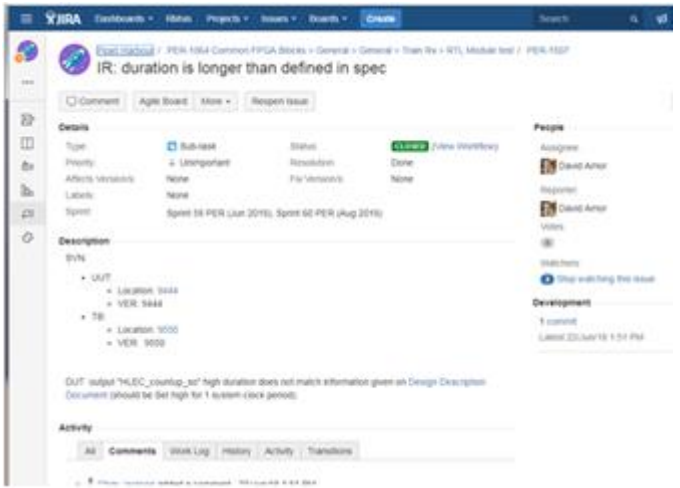
Integrated change tracking

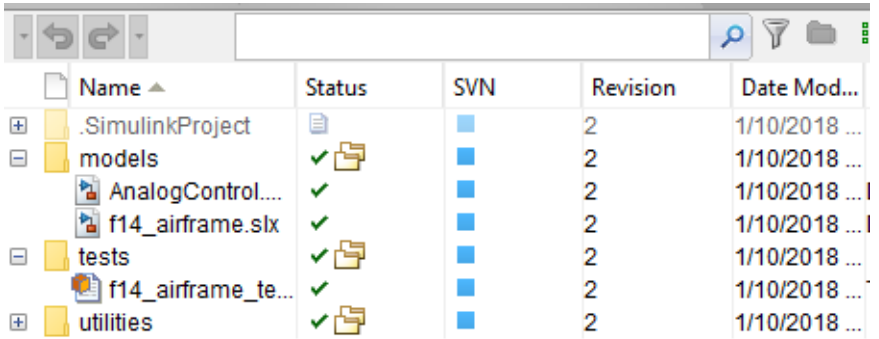
Simulink Projects:

Primarily projects enable correct path to reference model / scripts etc

Jira:

Jira is a task tracking system that can be integrated with SVN.





SVN (Subversion):

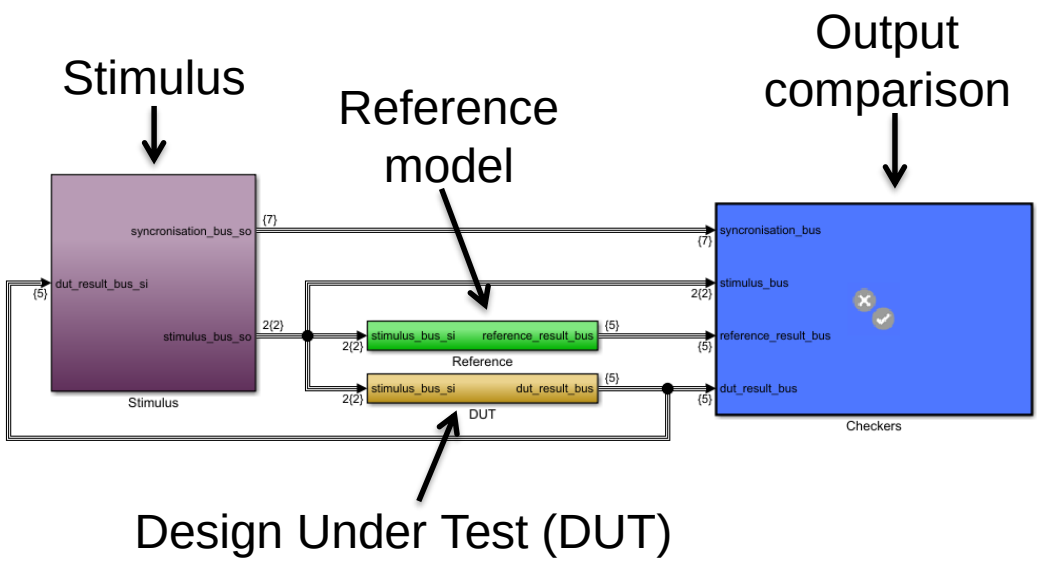
SVN is a versioning and change control system that is integrated with MATLAB.

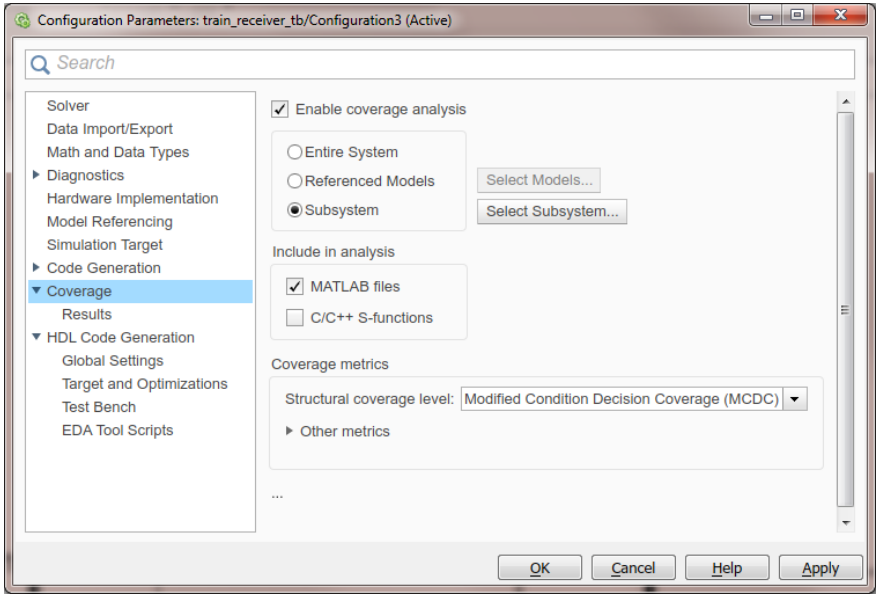
Test benching and Model coverage

Test metric

V&V (Verification and Validation)

Model coverage is a hint to code coverage - but quicker

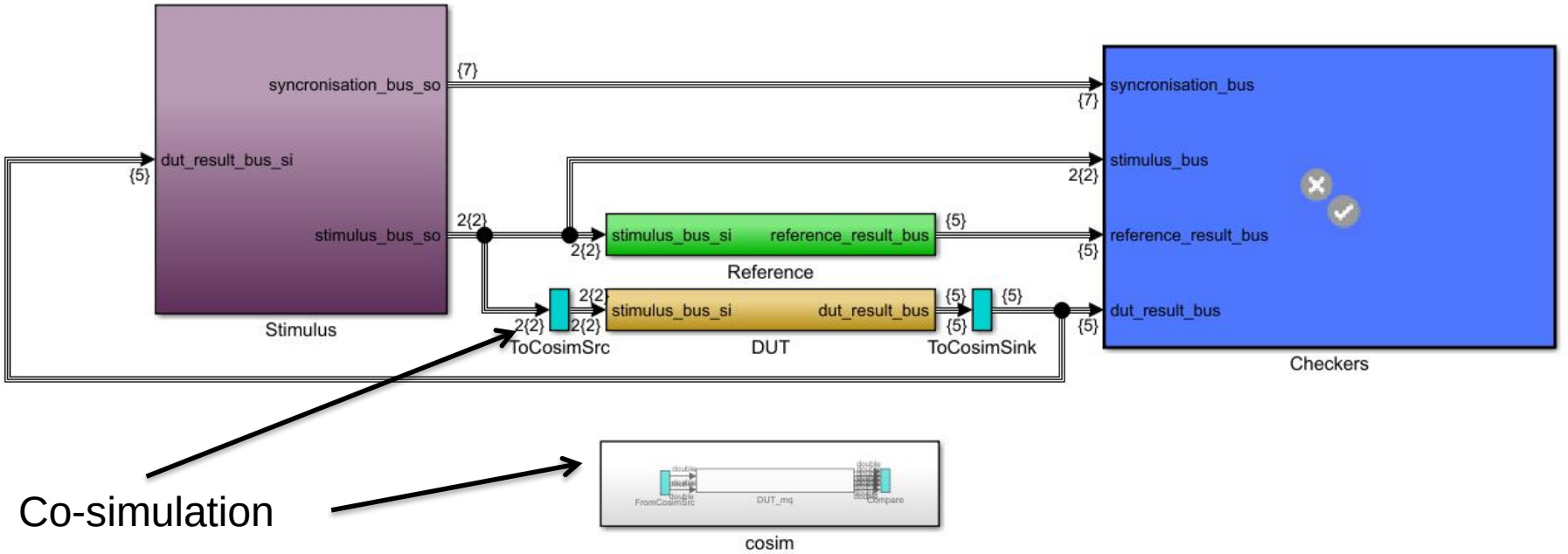




Co-simulation of generated HDL and Code coverage

Simulate generated HDL to confirm clock-by-clock equivalence of model.

Confirmation that VHDL = model & code coverage



Results: Future

Plans for Matlab/Simulink

- Rules based auto checking code to reduce code review time
- Investigate “continual integration” compatibility with Simulink
- Leverage toolboxes
  - Simulink test toolbox
  - Parallel toolbox
  - DSP toolbox
  - Unknown toolbox still under development…

Results: Challenges

with Matlab/Simulink

- Scopes – not designed for timing diagrams / logic
- Bidirectional port simulation
- Slow simulation – single threading (inherent)
- Functional debug requires systems engineer – shortcoming of model based design but also an advantage as this means more feedback on system level design.
- Recruiting engineers that have experience with HDL Coder.
- Single source design entry tool

Results: Benefits

of using Matlab/Simulink

- Consistent design-flow from conception to implementation using the same language.
- Reduced rework – Reduced misinterpretation & Unexpected emergent behaviour is observed earlier
- Its easier to update the FPGA and prove that the system requirements are still met.

Anecdotally:

- Extremely complex motor control system with almost no lab issues.
- Customer revision of requirements within months of project kick off.

Ultra Electronics | PMES

Questions

DEFENCE

SECURITY

TRANSPORT

ENERGY

Ultra ELECTRONICS