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<sup>®</sup>:

"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 **\$\$\$** Verification **Requirements**  Supporting functions - Design assurance for high integrity System design Verification systems - Long term support FPGA design Verification Implementation



# 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)



Not protectively marked

# 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...

Ultra ELECTRONICS

- 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

making a difference

- 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<sup>®</sup> 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





Not protectively marked

# 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



To next layer of MBD process



# Layer 4 – Functional block models

Layer 3: Control system model Set of functional block models created Motor System controller controller Interfaces between each functional block defined Sensor HMI interfaces Functional blocks tested Layer 4: Control System - Module Breakdown HMI Sensor Interfaces Motor Controller System Controller Madul Modulo Madula Modul Madula 2 Modulo 2 Modulo 2 Madula Module 1 Module 1 Module 1 Module 1



# 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: Embedded **M**ATLAB (EML)
- 4. Design: Stateflow
- **5**. Design: Using Buses
- 6. Design: Reuse Model References, libraries
- 7. Tool: Scripting (Hardware Description Language) HDL generation
- 8. Tool: Projects and change control with (**Subversion**) 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

		c_inv_mon_rtl_status - Simulink
		Edit View Display Diagram Simulation Analysis Code Tools
		- 🔄 🗢 🔶 🚼 🎯 - 🧱 - 📫 🔩 🕞 🕨 (
	Janation Parameters: un	ntitled/Conniguration (Active)
	2 Search	
	Solver	Simulation time
Ensure consistent	Data Import/Export Math and Data Types	Start time: 0.0 Stop time: 10.0
Results when co-simulating	<ul> <li>Diagnostics</li> <li>Hardware Implementation</li> </ul>	Solver selection
i toodito inien oo onnonating	Model Referencing	Type: Fixed-step    Solver: discrete (no continuous states)
	Code Generation	▼ Solver details
	Coverage HDL Code Generation	Fixed-step size (fundamental sample time): 1 / 100e6
		Tasking and sample time options
		Periodic sample time constraint: Unconstrained
		Treat each discrete rate as a separate task
		Allow tasks to execute concurrently on target
		Higher priority value indicates highs



Not protectively marked

# 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

- function cntr\_so = counter\_eml(end\_value\_si) Fixed point data types, Controlling data type bits 2 🖻 %#codegen % Counter An 8bit counter that terminates 3 The reshape "(:)" operator, Code of practice naming. % at "end\_value\_si" - reseting to 0 4 5 % Declare signals 6 7 persistent cntr\_s Ports and Data Manager (untitled/MATLAB Function 8 Edit Add Tools 9 % Reset ₽ fx - A A 💥 🔶 🛱 if isempty(cntr\_s) 10  $cntr_s = fi(0.0.8.0)$ : 11 -Name Scope Port Resolve MATLAB Function: MATLAB Function 12 end 🐻 u Input 1 Name: MATLAB Function fen 13 181 V Output 1 Update method: Inherited 

Sample Time: % Define outputs 14 MATLAB Eunction  $cntr_so = fi(0,0,8,0);$ 15 -Support variable-size arrays 16 Allow direct feedthrough % Assign outputs 17 Saturate on integer overflow cntr\_so(:) = cntr\_s; 18 -A E D C M T 4 3 Lock Editor 19 % The actual function Treat these inherited Simulink signal types as fi objects: Fixed: 20 🔶 Go To Diagram mulati if cntr\_s >= end\_value\_si 21 -MATLAB Function fimath 🗚 Edit Data Mode ew R 22  $cntr_s(:) = cntr_s + 1:$ Same as MATLAB Specify Other 23 else MULINK  $cntr_s(:) = 0;$ 24 hdlfimath() 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.

```
switch state s
49
50
51
            case 0 % awaiting start (incl intermessage gap)
52
                if start si == 1
53 -
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
                    busy s(:) = 1;
65 -
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:

50	Output 11	Inherit: Same as Simulink	unxi
<pre>     StartFrame_so </pre>	Output 12	Bus: StartFrame	StartFrame
PataFrame so	Output 13	Bus: DataFrame	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**



Ultra ELECTRONICS

making a difference

# Manual HDL generation **VHDL Output**

			Configuration Parameters: control/Co	onfiguration (Active)	×
Model Configuration parameters (cog "HDL Code Generation" -> Generate "HDL Code Generation" -> Generate "Industry of the second seco	y <u>r</u> am <u>S</u> ir	Configuration Parameters: control/Control/Control Control/Control Control Con	Image: Second standards       Asynchronous <ul> <li>Reset type:</li> <li>Asynchronous</li> <li>Reset asserted level:</li> <li>Active-nign</li> <li>Clock input port:</li> <li>Clock enable input port:</li> <li>Clock enable</li> <li>Clock edge:</li> <li>Rising</li> </ul> <li>Additional settings</li> <li>General Ports</li> <li>Coding style</li> <li>Coding standards</li> <li>Diagnostics</li> <li>Floating Point Target</li> <li>RTL Annotations</li> <li>Use Verilog 'timescale directives</li> <li>Inline VHDL configuration</li> <li>Concatenate type safe zeros</li>		
Code Generation Coverage HDL Code Generation Global Settings Target and Optimizations Test Bench EDA Tool Scripts	Generate HDL code Generate validation model Code generation report Generate traceability report Traceability style: Line Level * Generate resource utilization report Generate high-level timing critical path report Generate optimization report Generate model Web view Restore Model Defaults Run Compatibility Checker		Target and Optimizations Test Bench EDA Tool Scripts	Inline VHDL configuration Concatenate type safe zeros Emit time/date stamp in header Include requirements in block comments RTL Customizations   RTL Customizations   Represent constant values by aggregates   Inline MATLAB Function block code   Initialize all RAM blocks   RAM Architecture:   RAM with clock enable   No-reset registers initialization:   Generate initialization inside module   RTL Style   Use "rising edge/falling edge" style for registers	
	Generate			<u>OK</u> <u>Cancel</u> <u>H</u> elp <u>App</u>	ply



# 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	<pre>std_logic_vector(7 DOWNTO 0); uint8</pre>
cntr so	:	OUT	std logic vector(7 DOWNTO 0) uint8
);			
END counter eml;			
-			
ARCHITECTURE rtl OF coun	ter	eml IS	
	_		
Signals			
SIGNAL end value si unsigned		ed :	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



Not protectively marked

# Projects and change control with (Subversion) SVN with Jira Integrated change tracking

#### Simulink Projects:

Primarily projects enable correct path to reference model / scripts etc

#### <u>Jira:</u>

with SVN.

Jira is a task tracking system that can be integrated





#### 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



MATLAB Expo SLIDE 30

### 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



