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

MATLAB Expo 2018

Process Justin Lennox
FPGA David Amor
About Ultra Electronics
PMES

Justin Lennox
PMES scope of supply

Submarine example

- Electrolyser PSU
- Rod control gear
- Pressuriser heater controller
- Distribution system
- Motor and drive systems
- Control consoles
- EM signature management, Corrosion protection & Active shaft-grounding
- Lube oil inverter
- Power converters
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

Diagram:

- Requirements
- System design
- FPGA design
- Implementation
- Verification
- $$$
## Traditional design process

### Pros and cons

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

Wiki environment
- Functional and physical documentation trees
- Model and test bench documentation
- Interface definitions
- Engineering decisions
- FMEAs
- Design review records

Modelling environment
- MBD process
- Requirements verification
- Model flow-down
- Partitioning

Requirements management tool
- Customer requirements
- Derived requirements
- Route to verification
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.
FPGA
Development in a MathWorks Environment
With alignment to IEC61508

David Amor
## 10 years with MathWorks

### The mechanics of FPGA production

<table>
<thead>
<tr>
<th></th>
<th>Tool</th>
<th>Design</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Tool</td>
<td>Simulink fixed solver discrete step</td>
</tr>
<tr>
<td>2</td>
<td>Design</td>
<td>Schematics for Architecture</td>
</tr>
<tr>
<td>3</td>
<td>Design</td>
<td>Embedded MATLAB (EML)</td>
</tr>
<tr>
<td>4</td>
<td>Design</td>
<td>Stateflow</td>
</tr>
<tr>
<td>5</td>
<td>Design</td>
<td>Using Buses</td>
</tr>
<tr>
<td>6</td>
<td>Design</td>
<td>Reuse - Model References, libraries</td>
</tr>
<tr>
<td>7</td>
<td>Tool</td>
<td>Scripting (Hardware Description Language) HDL generation</td>
</tr>
<tr>
<td>8</td>
<td>Tool</td>
<td>Projects and change control with (Subversion) SVN with Jira</td>
</tr>
<tr>
<td>9</td>
<td>V&amp;V</td>
<td>Test benching and Model coverage</td>
</tr>
<tr>
<td>10</td>
<td>V&amp;V</td>
<td>Co-simulation of generated HDL and Code coverage</td>
</tr>
</tbody>
</table>
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.
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.

- 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:

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
Model Configuration parameters (cog)

“HDL Code Generation” -> Generate

- Automate the generation using a script that calls "makehdl"
- Consistent code output

VHDL Output

- Manual HDL generation
- VHDL Output
Comparison of Matlab code with VHDL output

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

ENTITY counter_cml 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_cml;

ARCHITECTURE rtl OF counter_cml 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); -- uint8
  SIGNAL cntr_s_next : unsigned(7 DOWNTO 0); -- uint8

END rtl;
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

- Stimulus
- Reference model
- Output comparison
- Design Under Test (DUT)
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
• It’s 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.