Adopting Formal Methods in the Verification of ARM Based CPU Subsystems

TVS Formal Verification Seminar

James Pascoe

23rd May 2013
• **Who are we:**
  - The ‘CPU’ part of ‘CPU/GPU’ in TR&D (ST Bristol)
  - We develop ARM based sub-systems for a range of SoCs

• **Organisation:**
  - System-level functional verification (Noida)
  - Block-level activities (Bristol)
  - Low-power and DFT verification (Grenoble)

• **Formal:**
  - Evaluated Jasper on three projects:
    - Sensor Control Block (SCB) – block-level verification
    - Clock and Reset Manager (CRM) – block-level verification
    - Documentation driven point-to-point connectivity checking
ESS = Event signal synchronizer
TSG = Time-stamp generator
RSE = Request source encoder
Verification Strategy

• ARM subsystem verification:
  • Unit testing:
    • Performed by Designers
    • Intended to answer the question: ‘is this block ready to enter verification?’
    • Designer typically implements HDL based test-benches that are not reused
  • Block-level:
    • Performed by Verification engineers
    • Answers the question: ‘does this block conform to its specification in isolation?’
    • ARM IP is considered verified 😊
    • Typically verified using a constrained random approach (Specman)
  • System-level testing:
    • Initial checks
      • Point-to-point: have we got the wiring basically correct?
      • Point-to-point: power awareness.
    • Functional testing
      • Does the subsystem function as a system? (Directed testing, Specman)
      • Does the subsystem give the correct level of performance? (EEMBC etc.)
Approach

• Formal projects were layered increasing in complexity:
  
  • **Sensor Control Block**
    • Digital memory mapped sensor block
    • Developed in Bristol (design team is on-site)
    • Well specified and understood
    • Constrained Random test-bench developed in parallel
  
  • **Clock and Reset Manager**
    • Provides clock and reset sequencing in subsystem (complex)
    • Developed in Grenoble
    • Very good micro-architectural documentation but no functional specification
    • Constrained Random test-bench developed in parallel
  
  • **Point-to-point connectivity checking**
    • Flow developed to extract assertions from project specifications
    • Use Jasper to verify connectivity at the subsystem level
    • Supersedes old in-house flow
    • Also very useful in the context of low-power (e.g. UPF aware proofs)
Sensor Control Block

• Provides a digital front-end to thermal sensors:
  • Monitors chip temperature and generates interrupts
  • Selected for its simplicity:
    • APB accessed memory mapped register file
    • Connects to thermal sensors
    • Samples sensors periodically
    • Generates an interrupt when over threshold
  • Well specified and understood

• Used Jasper to check:
  • Register interface
  • APB interface and protocol
  • Min, max and averaging functionality
  • Sampling periods
  • Interrupt properties
Results

• Found 3 bugs:
  • Found subtle problem with PREADY signal on APB interface
    • Not detected by the Specman TB
  • 1 RTL problem:
    • Inverted ‘or’ concatenation for sensor overflow / underflow bits
  • 1 specification problem:
    • Registers denoted as ‘Write Clear 1’ can be cleared with other write operations

• Validity:
  • Didn’t expect to find much wrong 😊
    • Designer had performed extensive unit testing prior to verification
    • Some previous Specman verification performed
    • Tried and tested components used in the development
• We need to answer the following questions:
  • How complete is the verification?
    • Question: are there enough assertions to verify all features?
    • Answer: measure out-of-coi coverage to find coverage holes
  • Is the environment over constrained?
    • Question: are we masking bugs by constraining the environment too much?
    • Answer: measure stimuli coverage to check that all legal scenarios are covered
  • How good are the bounded proofs for my block?
    • Question: do we require more depth in the search space?
    • Answer: use bounded coverage analysis
  • How well does the combination of formal and dynamic cover the design?
    • Question: how do I interpret formal coverage alongside constrained random coverage?
    • Answer: Jasper is working on merging results into other UCDB files
Clock and Reset Manager
Clock and Reset Manager

• Generates clocks and resets to blocks within subsystem:
  • Sequences actions to control transitions between operating points
    • Enter low-power mode, exit from retention mode …
  • Enables subsystem to be fully asynchronous
  • Provides an abstract interface to the SoC
  • Has to conform to ARM sign-off criteria
  • Good micro-architectural documentation but difficult to verify against 😊
  • Critical block and complex

• Verification approach:
  • Used Jasper to perform feature extraction
    • Easier than using program based approaches
  • Proved critical properties using Jasper
    • Event sequences, specific timings etc.
  • Developed a Specman test bench in parallel
    • Compare approaches
Feature Extraction

• Key problem: CRM functionality not well specified:
  • Once ‘cmd_in’ rises ‘cmd_ack’ should follow after ‘N’ cycles
  • Once ‘cmd_in’ falls, ‘cmd_ack’ will follow after ‘M’ cycles
  • where ‘N’ and ‘M’ are not known …

• Solution
  • Use Visualise to evaluate the range of cycles the ‘ack’ follows
  • Write number of properties with different values for ‘N’
  • Find the lowest ‘N’ for which the property proves
  • Keep the assertion of the lowest proven ‘N’
    assert: $\text{rose}(\text{cmd} \_ \text{in}) \mid \rightarrow \#\#[1:80] \text{cmd} \_ \text{ack}$
  • Define cover item for the Min ‘N-1’ for regression
    cover: $\text{rose}(\text{cmd} \_ \text{in}) \mid \rightarrow \#\#1 !\text{cmd} \_ \text{ack} [*80-1]$
Point-to-Point Connectivity Checking
Point-to-Point Connectivity Checking

• Point-to-point connectivity checking:
  • Once everything is verified at the unit / block-level …
  • Point-to-point connectivity checking provides a first check that blocks have been assembled into the subsystem correctly
  • Eliminates wiring errors; useful before functional system testing

• Developed documentation driven point-to-point flow:
  • 2564 reference connections generated from key project document
  • Point-to-point checking flow setup in 1 day (with help of Jasper)
  • Was able to prove 2511 properties in the first week
  • All properties were proven in two weeks

• Useful for low-power:
  • Can check connectivity rules for changing power states
  • Rule validity can be tied to power states
Conclusions
How to Sell Formal To Your Manager

• Sources of Added Value:
  • Tangible effort savings stem from not having to build large test-benches
    • Seems to take ST 6 m/w to build a block-level test-bench
  • Properties can be added incrementally and reused throughout the block’s lifetime
    • Capturing previously wasted effort by embedding properties in the RTL
  • Useful for reasoning about designs that are not fully understood
    • Is responsive enough to be done in real-time
  • Point-to-point testing was a low-effort, high-value activity
    • Provides a basic sanity check of the wiring

• Deployment:
  • Designers like using it – provides rapid high quality feedback
    • Easier to learn than programming language solutions
  • Build a group of internal ‘champions’
    • More successful than trying to educate the masses
  • Integrates easily into project management methodologies
    • Constraint coverage and progress can be checked easily
Questions