Atmel and the use of Verilator to create uC Device Models

Dag Braend
Sr Director Atmel MCU Tools

Roland Kruse
Jie Xu
Jan Egil Ruud
Atmel Co-simulation Team

Atmel MCU Tools

Background

- Atmel is a major provider of 8 and 32 bit MCU solutions
  - AVR proprietary architecture for consumer and industrial 8 bit devices
  - ARM Cortex M4 architecture for consumer and industrial 32 bit devices
  - maXTouch proprietary architecture for capacitive display touch applications
  - Atmel specializes in picoPower technologies – latest offering is the lowest power ARM Cortex M4 based device – SAM4L
- Atmel MCU Tools is the primary marketing channel for our devices
  - Atmel Studio
    - Complete Integrated Development Platform
  - Atmel Gallery
    - Atmel and 3rd party functional extensions
  - Atmel Spaces
    - Collaborative workspace with Atmel Studio integration

Atmel MCU Tools

Simulator

- Atmel uses MCU simulators as a debugging tool for customers doing application development
  - We currently do NOT use simulators as a way to validate the IC development
  - We do NOT use simulators in a systems simulations fashion (why? They are too slow)
  - We use simulators as a target where the customers can validate their C/C++ MCU code before they deploy the code on an actual HW target
  - Simulators are integrated as targets in the Atmel Studio IDE
  - The IDE does not see the difference between a simulated target and an actual HW target – they should behave the same

- For this scenario to work we need
  - Cycle accurate simulation of the complete device, including different memory types and analog front-ends.
  - Only way to do this is to re-use the code for the actual device when making the simulator model
  - Atmel currently sells > 350 uC models. We need a simulator model for each. Thus the model creation process is extremely important

Atmel MCU Tools

Simulators - History

- Atmel has worked with creating simulator models for many years and has employed different tool chains for the job
  - VTOC
  - Carbon Design Systems

- The use of commercial tool chains has proven difficult
  - This is a complex and specialized field with not so many customers
    - = expensive tools
  - The companies that support the tools tend to be small
    - = will support their major key customer at any given time but if you are not the largest you suffer
  - Some of the providers have business models and license agreements that does not fit well with free distribution of unlimited number of licenses
    - Some of the providers have business models that makes Atmel a competitor of the company providing the technology
  - Conclusion: Atmel wanted to explore the use of open source tools for this development

Atmel MCU Tools

Main advantages of working with open source tools

- No license fees.
  - What we pay goes into real work, either development of new features, or real support. If we choose to not pay for these services anymore, we still have the tools.
- Tools will always be available, no matter what happens to the project.
  - Both our earlier changes of tools were enforced by non-technical issues with the companies owning the tools.
- No concerns with software license management
  - Atmel can deploy the tool chain at multiple design locations without difficult license negotiations
- We benefit from work contributed by others,
  - at no additional cost.
- Source code access
  - Having the source code of the tool is very helpful when problems occur, or when we create in-house tools to support our workflow.
- Reduced bug fixing cycle time
  - The open source model have the potential of drastically shortens the cycle when bugs are fixed, we don't have to wait until the vendor produces a new version (if ever).
  - But it requires an active community and a certain level of use

Atmel MCU Tools

The psychology of SW purchases

- Companies (read Atmel) are less willing to buy expensive SW
- SW license management is always a hassle – and multinational companies like Atmel don't like it when you start talking of "site" licenses, licenses located to geographies, encounter different pricing schemes in different regions
- Companies are however willing to pay consultant fees in order to have required SW modifications done
- An open source model where companies (like Atmel)

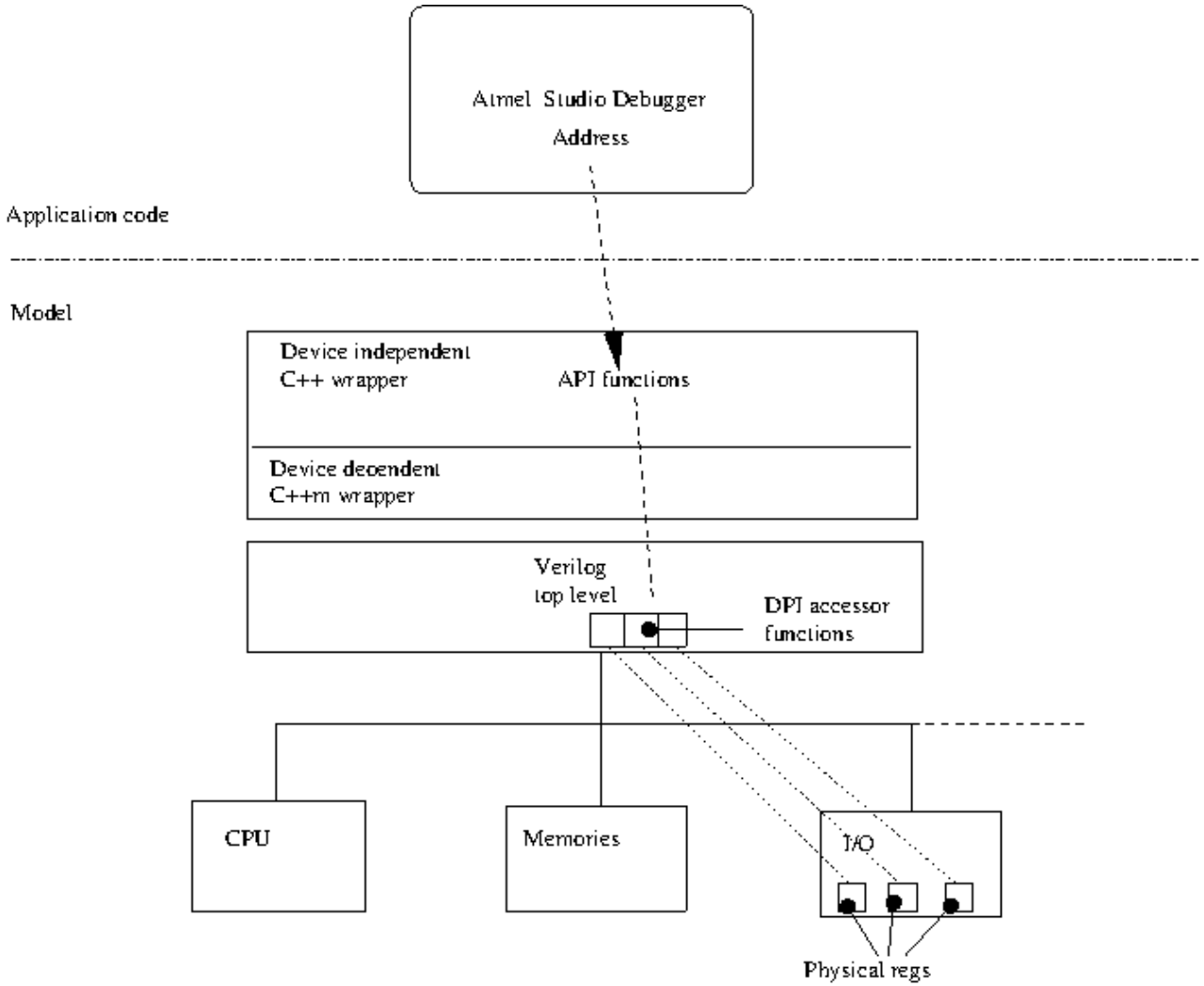
Atmel MCU Tools

Simulator

- Atmel and Verilator
  - Atmel has visited the Verilator project on several occasions over the years and previously the feel was that the tool was immature and would not fit Atmel's needs.
  - Atmel revisited the tool in 2011 and found that the situation had changed – the tool seemed much improved
  - In 2012 Atmel engaged with Embecosm in order to make Verilator support the required Atmel device implementation and development flow.
  - There has been a number of technical issues to be resolved, and in addition Embecosm has implemented System Verilog support.
  - The experience with getting Verilator to work has been comparable to the commercial tools – there is no "out of the box" experience to be had in this respect

Verilator work flow

Simplified view of the simulator model architecture.



Verilator work flow

Main steps of model creation

- Replace top level module. Remove non-synthesizable parts of the hierarchy. Adapt top level I/O ports to fit C++ wrapper.
- Create replacements for modules that cannot be used with Verilator. This includes NVM, UDPs, etc.
- Make the design compile with Verilator. This may require minor changes in RTL code throughout the hierarchy.
- Create DPI functions to access memories, CPU registers, and I/O from C++ code. This process is partially based on machine-generating Verilog and C++ code from an XML description.
- Write the device-dependent part of the C++ wrapper. This part is responsible for driving the model (reset, clocks, etc).
- Test the completed model.
- Set up for production build. This includes creation of the Windows version of the model (cross-compiled on Linux)
- The model is tied to Atmel Studio with a small XML file (binding device name to DLL file name)