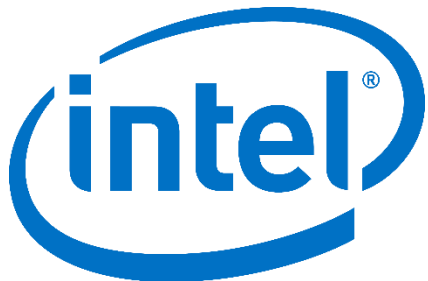


Better and Faster Virtual Platform (VP) Modeling using a Domain-Specific Language (DSL)

Jakob Engblom, jakob.engblom@intel.com

Erik Carstensen, erik.carstensen@intel.com

Intel, Stockholm, Sweden



About us

- Jakob Engblom

- Product management engineer and simulation evangelist
- Working with Intel® Simics® virtual platforms since 2002
- PhD from Uppsala University

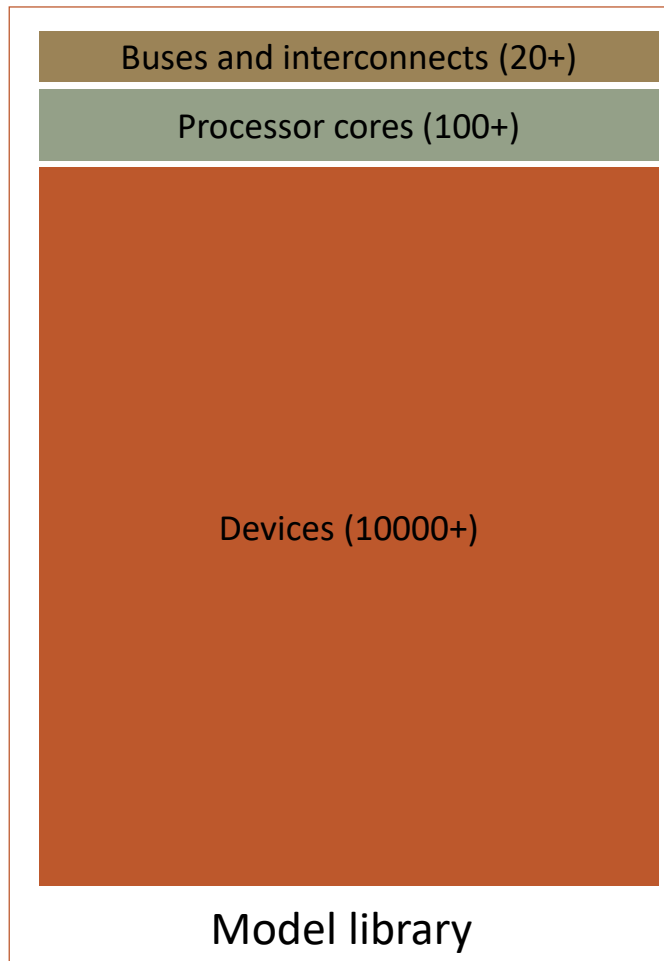


- Erik Carstensen

- Senior SW engineer, lead developer of the DML compiler
- Simics developer since 2006
- MSc from Uppsala University



Device Modeling Dominates VP Creation



- Virtual platform (VP) value is higher the earlier the platform is available for use – faster is better
- Device models dominate the virtual platform models
 - Many
 - Varied
 - Rapid change
- The major problem is **efficiently building device models** (for the VP modeling organization)
 - Minimum work, minimum time



Speeding up Modeling with Simics® DML

The Simics® **Device Modeling Language** (DML) is a **domain-specific language** (DSL) for device models for **fast functional** virtual platforms

- First version released in 2005, second revision in 2007, third in 2019
- Before DML, we used the Simics API from C code for modeling



The Design of Simics[®] DML

- **Domain-specific language == standard way to increase productivity**
 - *Strictly better than a modeling library embedded in a general language*
 - *Domain: fast functional models (also known as transaction-level modeling, or TLM)*
- Goal for DML design:
 - Make it hard to write bad models
 - Readable and maintainable code
 - Good target for code generators
 - DML compiler combines different parts of the model from different files
 - Automatically generate support for simulator platform features
 - Such as checkpointing, instrumentation, register metadata, ...
- DML generates C code with Simics API calls



What it Looks Like

- Provide natural modeling constructs
 - Banks, registers, bit fields, ...
 - Connects, interfaces, ...
 - Attributes, data, ...
 - Arrays and groups of objects
- Expressive template mechanisms
 - **Standard library** for common behavior
 - **Templates** for common device classes
 - **Custom template** libraries for specific models or families of models

```
// Partial example of as device model
dml 1.4;

device sample_i2c_device;

import "simics/devs/i2c.dml"; // generic i2c
import "platform-i2c.dml"; // i2c logic shared with other platforms
import "fuse-common.dml"; // common platform fuse mechanisms

// generated code with register declarations
import "DevBank_gen_code.dml";

// instantiate the register bank from the file
bank regs is i2c_ctrl_reg_bank {
  register hst_cnt { // Added manual code
    method write_action() {
      if (START.get() != 0) {
        START.set(0);
        send_start();
      }
    }
  }
}

// Generated file DevBank_gen_code.dml
dml 1.4;
import "access_templates_14.dml";

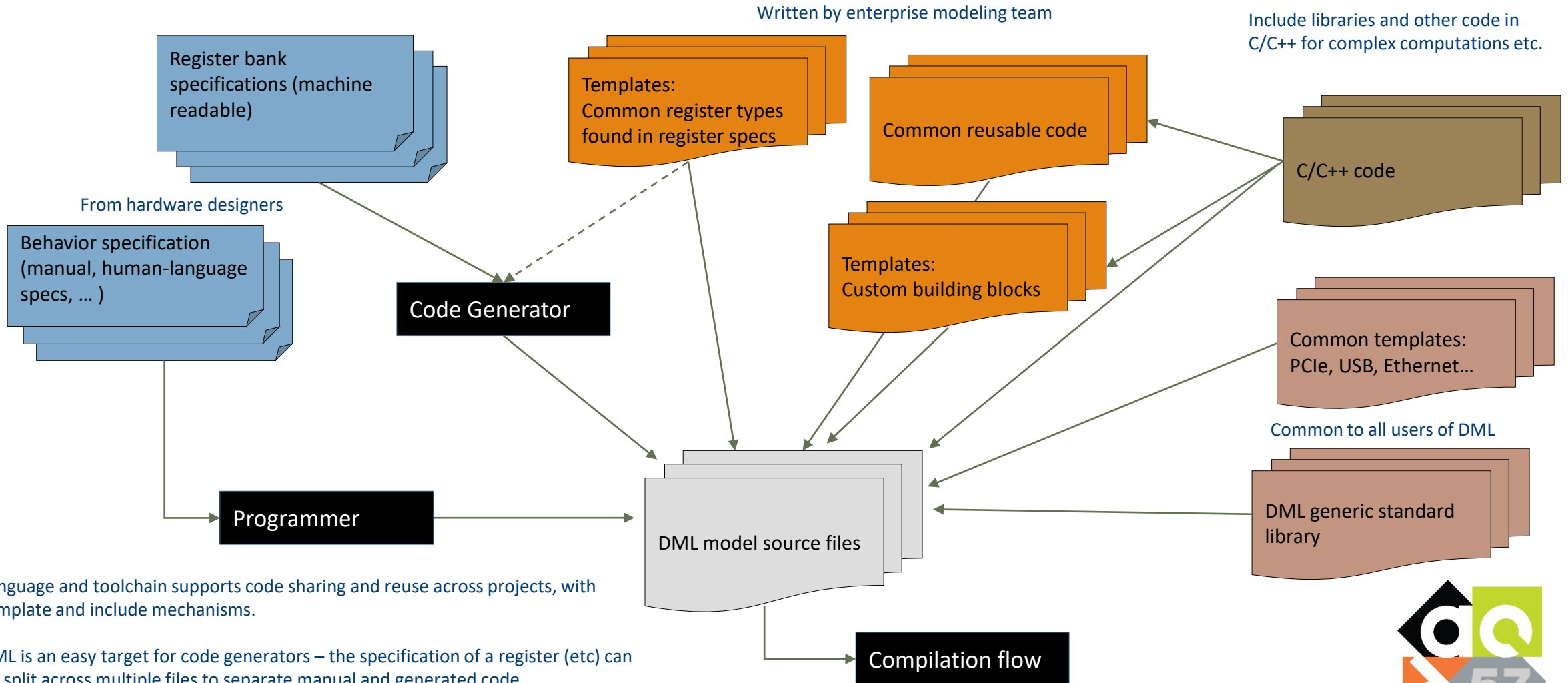
template i2c_ctrl_reg_bank {
  param bank_reset_signal;
  param register_size = 2;

  register hst_sts @ 0x00 "Host Status";
  register hst_cnt @ 0x02 "Host Control";
  // array of registers
  register tx[i < 8] @ 0x08 + i "Transmit data";

  // flesh out fields in hst_sts
  register hst_sts {
    field BYTE_DONE_STS @ [7:7] is write_1_clears;
    field INUSE_STS @ [6:6] is write_1_clears;
    ...
  }
}
```



Enterprise-Level Modeling: Reuse & Code Gen



Language and toolchain supports code sharing and reuse across projects, with template and include mechanisms.

DML is an easy target for code generators – the specification of a register (etc) can be split across multiple files to separate manual and generated code.



DML Works – Observed Benefits

- Proven in use for 15 years
- 50% to 80% smaller code than if written in plain C (using the Simics API)
- Faster device modeling than using other approaches (by experience)
- Small teams can manage dozens of 1000+ device model platforms
- Scales to devices with 100000s of registers and complex behaviors
- New developers get started within a **day**, useful within a **quarter**, **independent** within six months
- New Simics® features can be supported by just changing the code generator (*higher semantic level than libraries or API*)




```
957 //-----
958 //
959 //
960 // MSI-X interrupts
961 // - Driver software needs to configure how to send each interrupt in the table,
962 //   configuring message data and message address
963 // - This all just works automatically with the Linux general PCIe code
964 //
965 //-----
966 bank dev_msix_table is (msix_table_bank,
967                       function_mapped_bank) {
968     param desc = "PCIe MSI-X Table";
969     param function
970       = 2; // used to connect to the PCIe BARs
971     param num_of_vectors = MSIX_VECTORS;
972 }
973 bank dev_msix_pba is (msix_pba_bank,
974                    function_mapped_bank) {
975     param desc = "PCIe MSI-X Pending Bit Array";
976     param function
977       = 3; // used to connect to the PCIe BARs
978     param num_of_vectors = MSIX_VECTORS;
979 }
980 //-----
981 //
982 //
983 // Control register bank declaration
984 // - Just names and offsets, no functionality
985 //
986 //-----
987 bank regs is (function_mapped_bank) {
988     param desc = "LED controller main control register bank";
989     param documentation = "Registers used to control the device operation." +
990       " The main HW/SW interface of the device.";
991     param function
992       = 1; // used to connect to the PCIe BARs
993     param register_size = 4;
994 //
995 // Registers controlling the device itself - or rather, the main processor
996 //
997 register enable @ 0x0000 "Enable device operation";
998 register reset @ 0x0004 "Reset the device";
999 //
1000 // Registers controlling the display and display status
1001 //
1002 register update_display_request @ 0x0010 "Update of display requested";
1003 register update_display_status @ 0x0014 "Current status of display update operation";
1004 register update_display_interrupt_control @ 0x0018 "Interrupt control for display updates";
1005 //
1006 // Framebuffer
1007 //
1008 register framebuffer_base_address @ 0x001c "Base address of the framebuffer in local memory";
1009 //
1010 // Advanced display functions that firmware would implement
```

Questions?

And come see us in the virtual poster session!



Legal Notice

No license (express or implied, by estoppel or otherwise) to any intellectual property rights is granted by this document.

Intel disclaims all express and implied warranties, including without limitation, the implied warranties of merchantability, fitness for a particular purpose, and non-infringement, as well as any warranty arising from course of performance, course of dealing, or usage in trade.

Intel, the Intel logo, and Simics are trademarks of Intel Corporation or its subsidiaries in the U.S. and/or other countries.

*Other names and brands may be claimed as the property of others

© Intel Corporation.



END

