Firmware Firmly under Control: New Optimization and Verification Techniques for Application Specific Electronic Systems

Daniel Große (Univ. Bremen / DFKI)
Manuel Strobel (Univ. of Stuttgart)
Daniel Mueller-Gritschneder (TU Munich)
Vladimir Herdt (Univ. of Bremen)
Tobias Ludwig (Univ. of Kaiserslautern)

This contribution is funded as part of the CONFIRM project (project label 16ES0564-70) within the research program ICT 2020 by the German Federal Ministry of Education and Research (BMBF) and supported by the industrial partners Infineon Technologies AG, Robert Bosch GmbH, Intel Deutschland AG, and Mentor Graphics GmbH.

Outline

1. Motivation
2. Design-Time Optimization Techniques for Low-Power Embedded Memory Subsystems
3. Automatic HW-SW-Interface Generation and Optimization
4. RISC-V based Virtual Prototype for Efficient Simulation of Firmware-based Designs
5. Properties-First Design: A New Design Methodology for SoC Hardware and Low-Level Software

Motivation

Application-specific adaptivity ⇨ **Firmware**

Automotive

Power Control

Smartphone

Focus

New methods for early, efficient and systematic firmware design taking the underlying hardware architecture consisting of digital and analog components into account

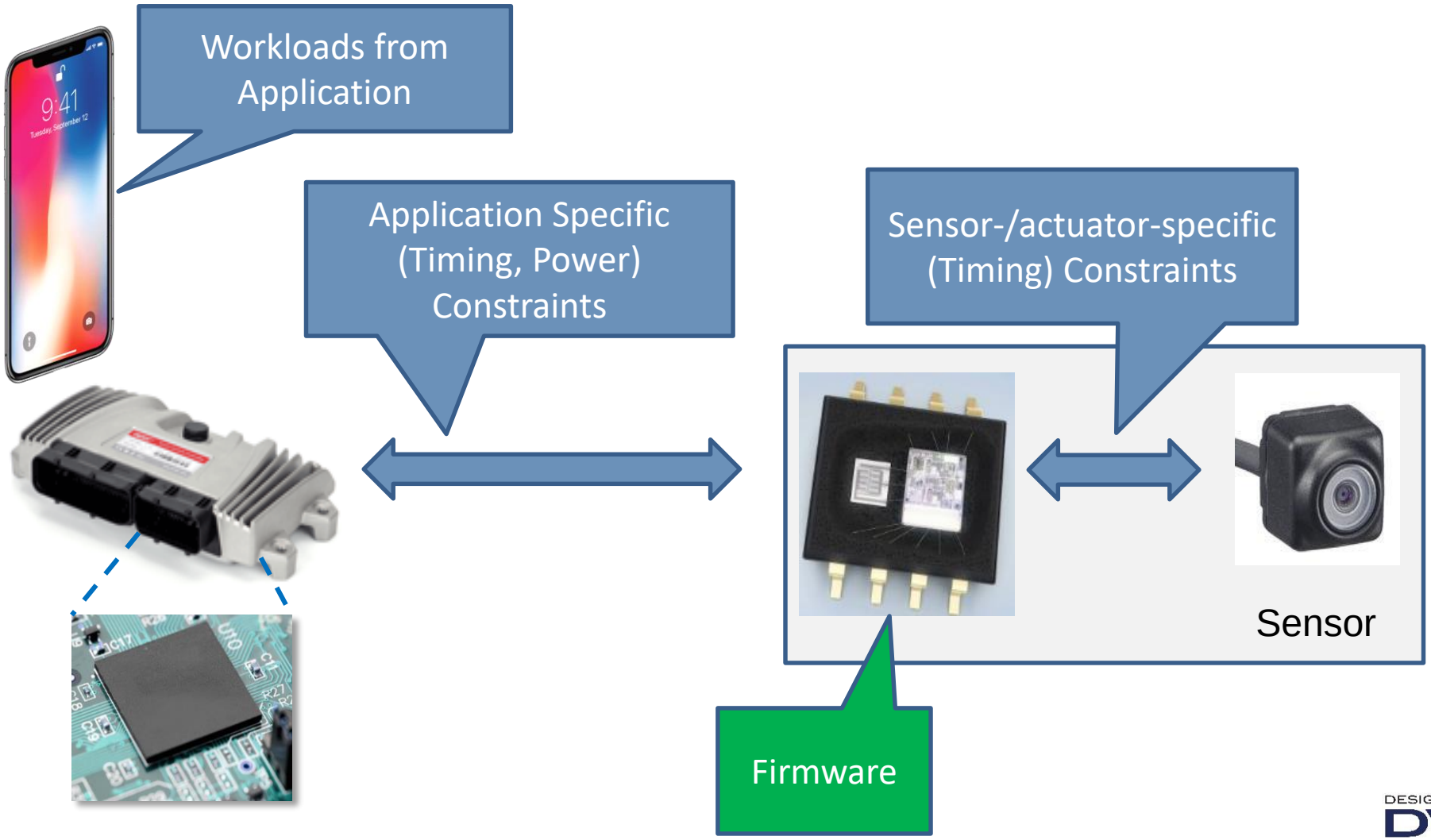
Automotive

Power Control

timing and power budgets

Smartphone

Action Chain: Sensor subsystem ↔ device



Abstraction Levels in Firmware Design

Firmware Levels

Application code/ Use Cases (HW independent)

Workloads, Constraints, Platforms

Hardware-dependent data processing

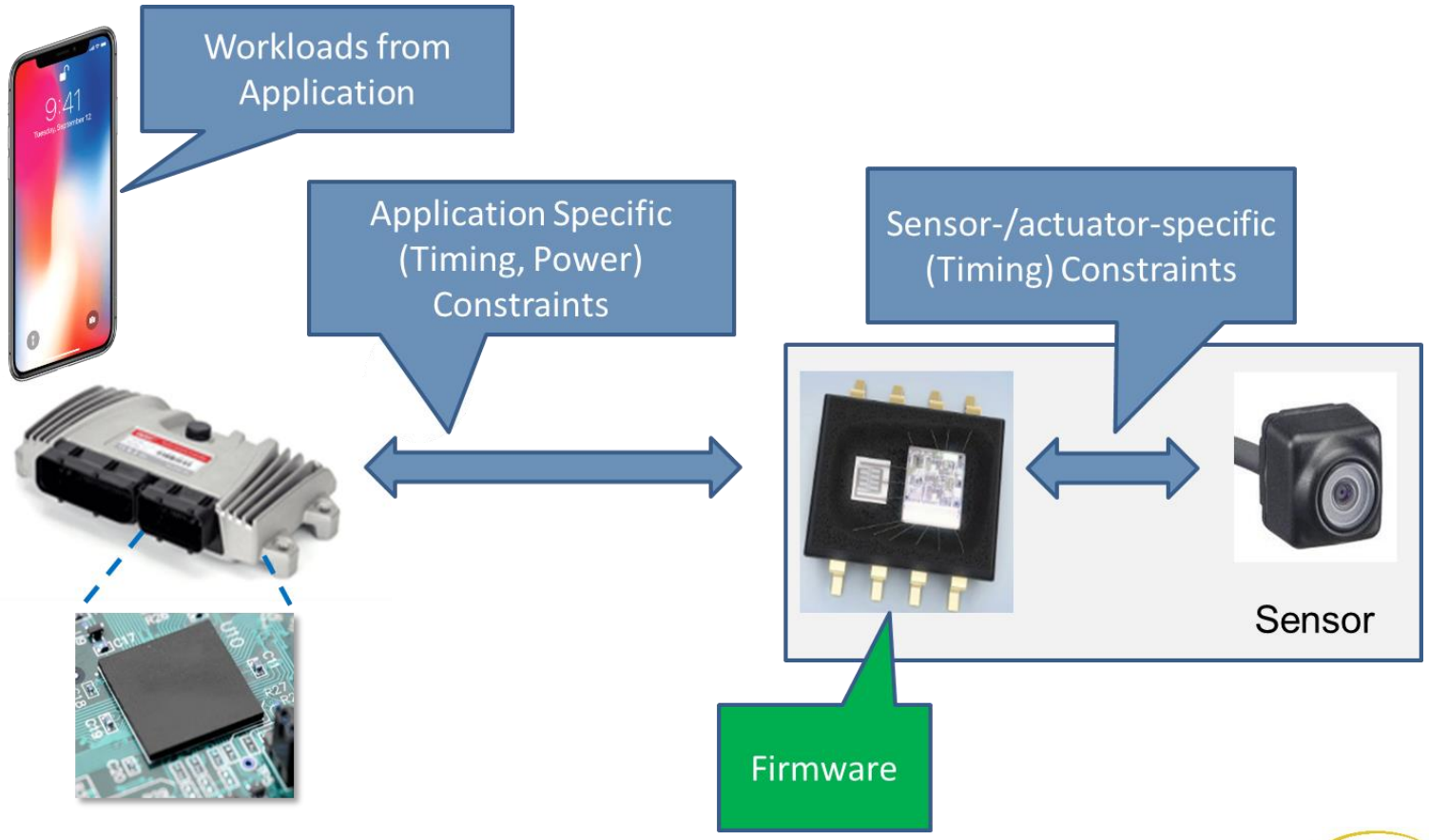
Edge Processing, Sensor Pre-processing,…

Runtime environment

Bare-metal, Scheduler, RTOS, …

Driver

Device Driver, HAL, …



Abstraction Levels in Firmware Design

Firmware Levels

Application code/
Use Cases (HW independent)

Workloads, Constraints, Platforms

Hardware-dependent
data processing

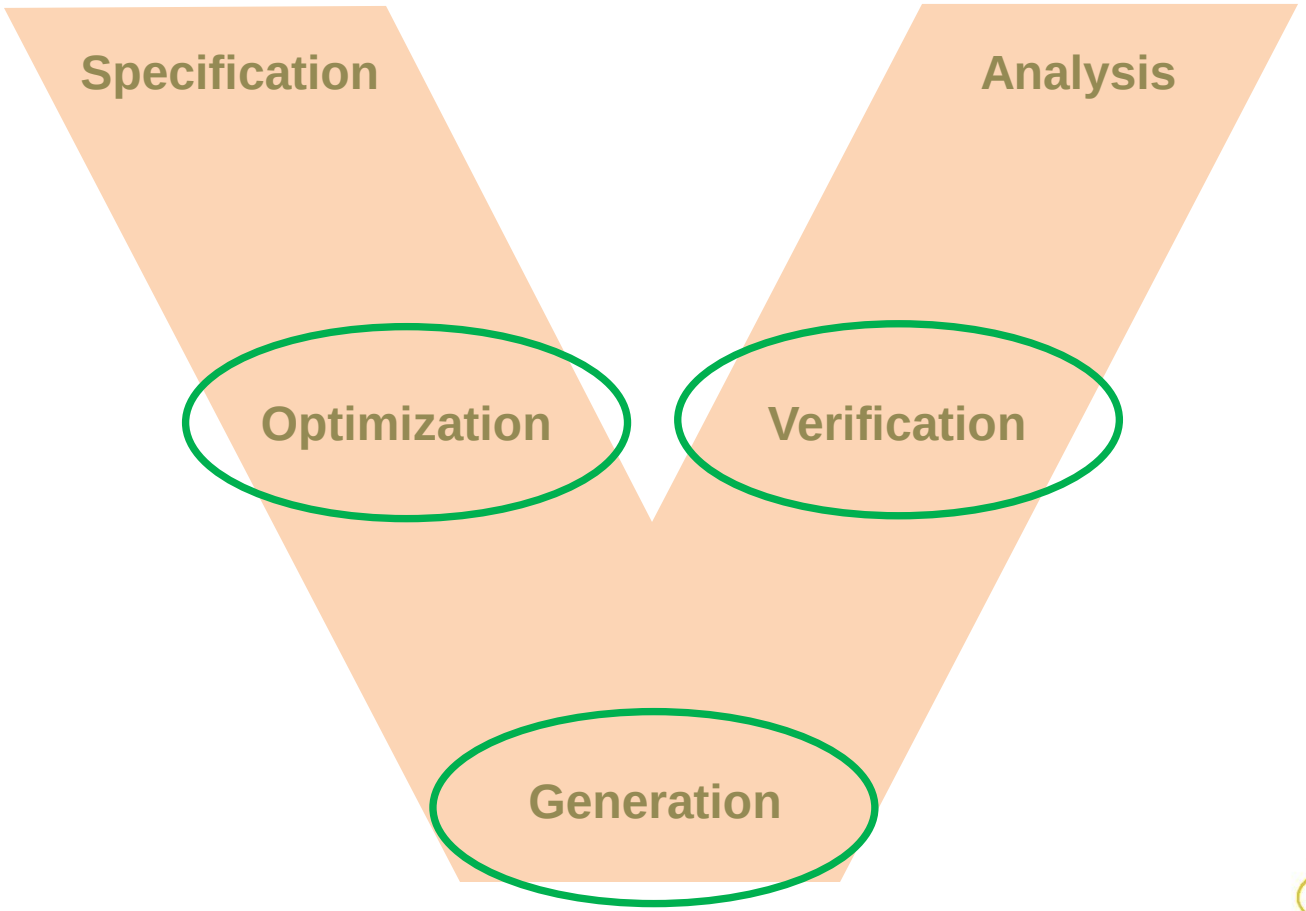
Edge Processing, Sensor Pre-processing,…

Runtime environment

Bare-metal, Scheduler, RTOS, …

Driver

Device Driver, HAL, …



Design-Time Optimization Techniques for Low-Power Embedded Memory Subsystems

Manuel Strobel (Univ. of Stuttgart)

accellera SYSTEMS INITIATIVE

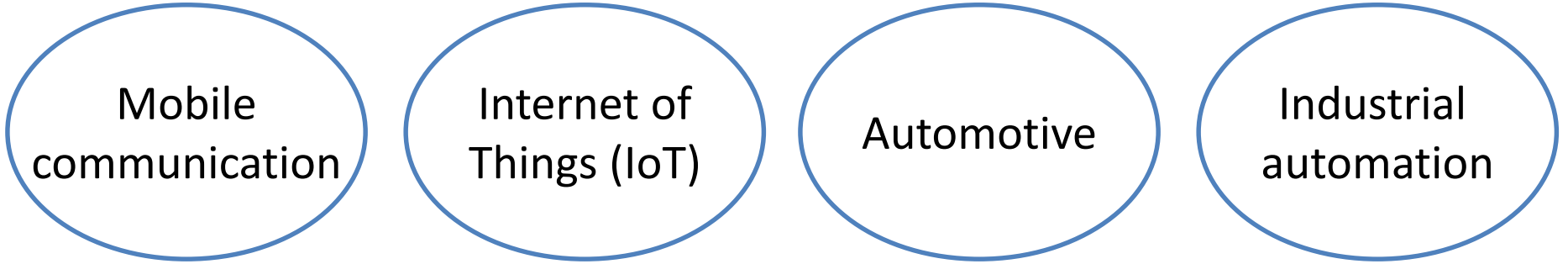
2018 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION EUROPE

Content

- Introduction
  - Motivation
  - Background
- Reduction of Dynamic Energy Consumption
  - Optimization Concept
  - Workflow
- Reduction of Static Energy Consumption
  - Optimization Concept
  - Workflow and Example
- Conclusion

General Motivation

- Energy and power consumption of System-on-Chip (SoC) devices is of vast importance in many application fields, e.g.:



Reduce energy consumption

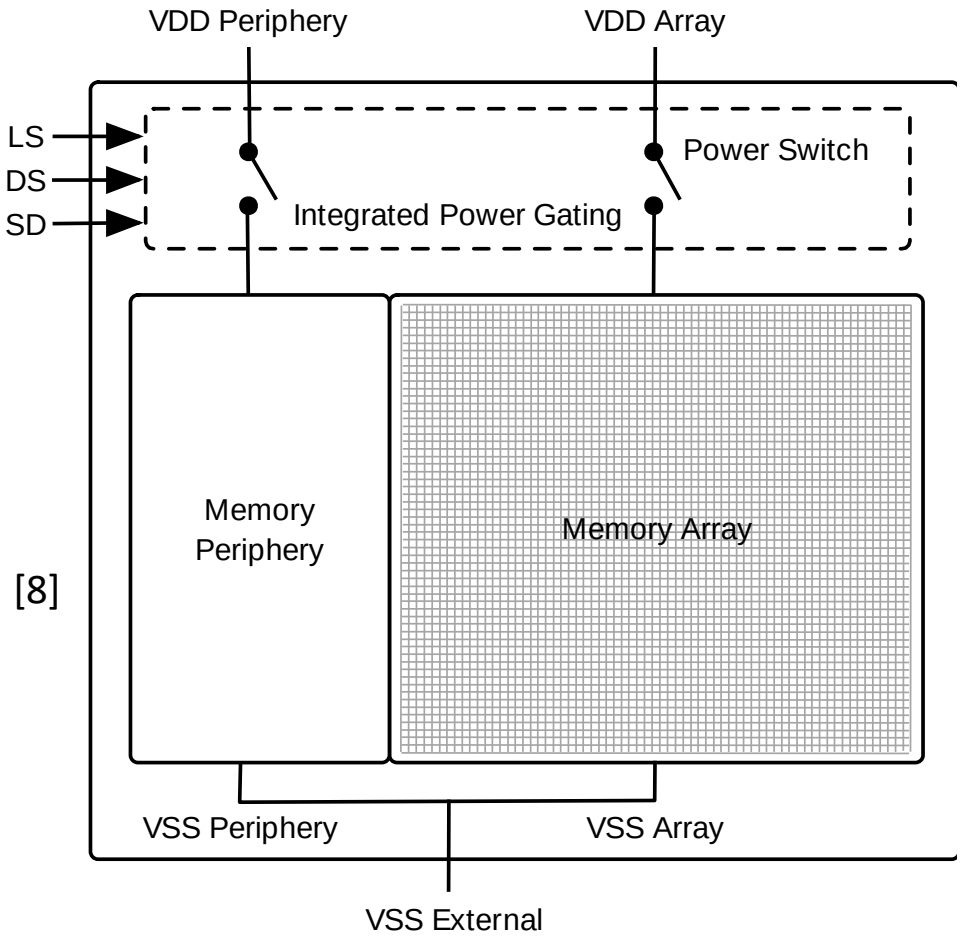
- Battery life
- Reliability

Control peak power [1]

- Packaging
- Heat dissipation
- Side effects in mixed-signal designs

SRAM Optimization Potential

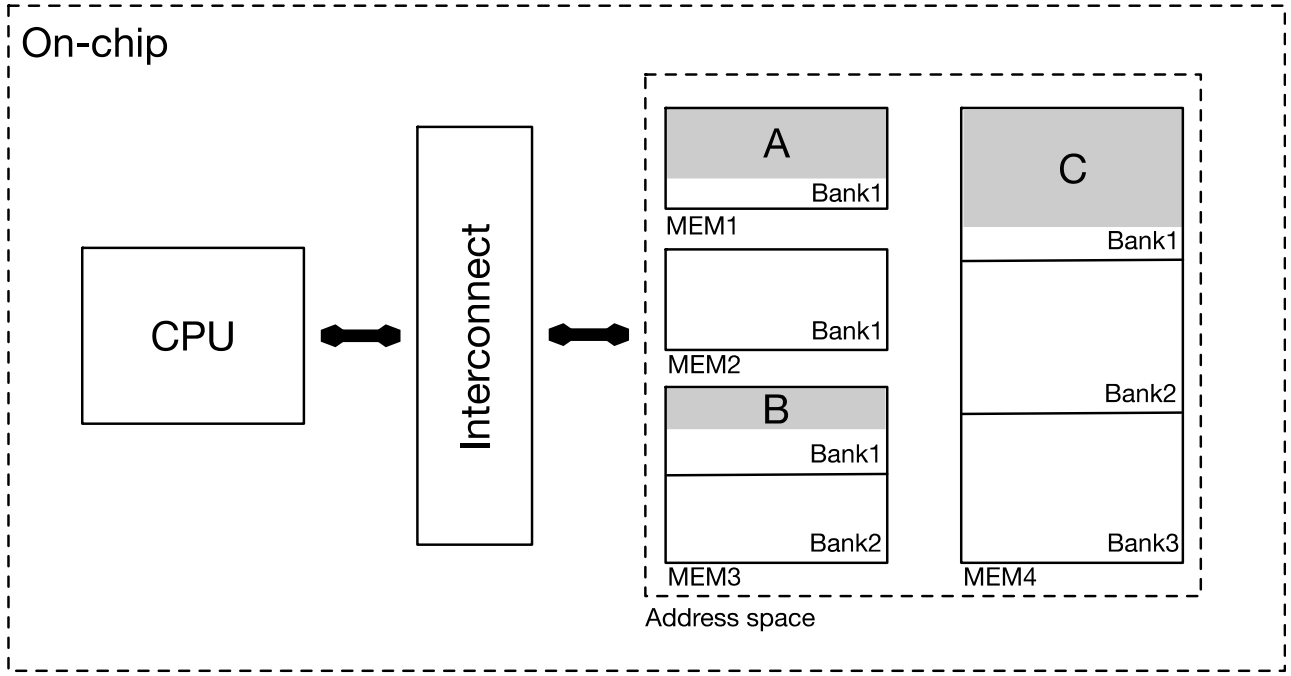
- Memory subsystem consumes a large part of the system energy budget
  - In memory intensive applications up to 75% [2]
  - Promising target for optimization
- Saving potential of SRAM originates from
  - Relation of memory size ⟷ energy consumption [8]
  - Low-power modes
    - Light Sleep (LS) – Biasing techniques
    - Deep Sleep (DS) – Power gating of periphery
    - Shut Down (SD) – Power gating of periphery & array



Based on [3] Fig.2 and 4

Optimization Variables

- Heterogeneous memory subsystem (SRAM)
  - Flat memory hierarchy
  - No monolithic memory block
- Optimization variables
  - Allocation (memories)
  - Binding (application)
  - Scheduling of low-power modes (if available)

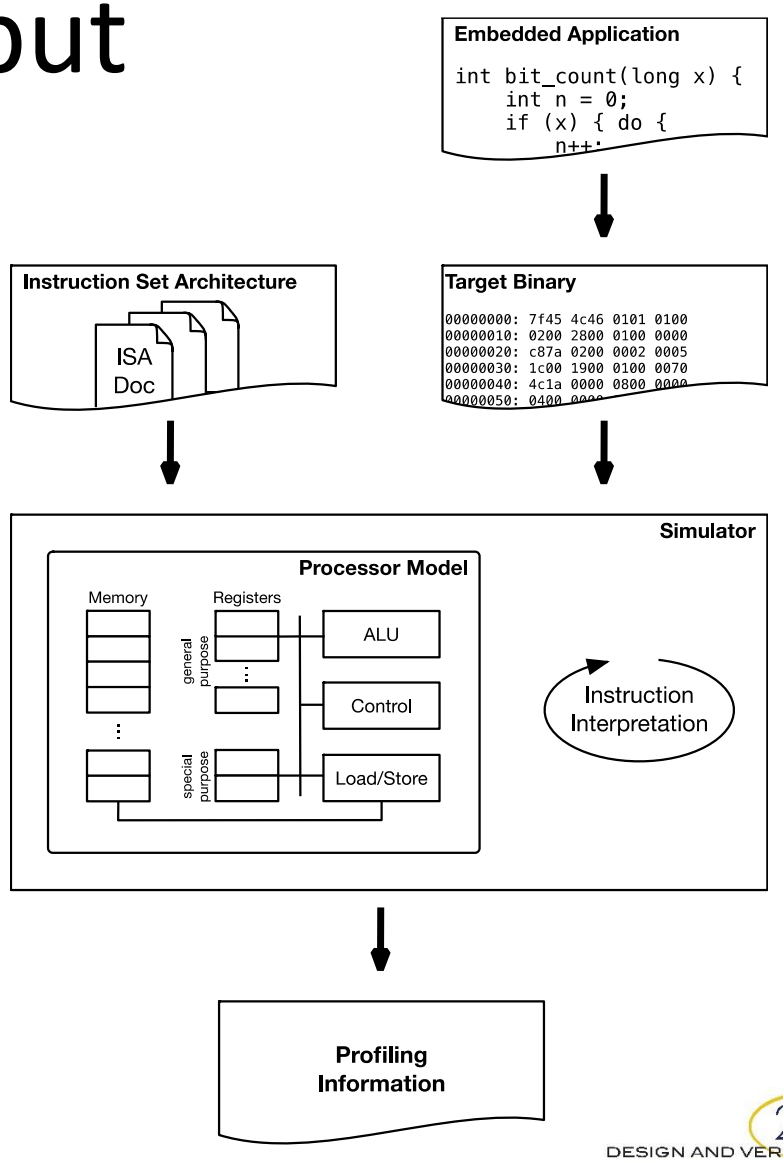


Based on [4] Fig. 2

Optimization Input

- Optimization based on profiling information
  - Generated from instrumented instruction set simulation (ISS)
  - Provides memory access statistics
  - Dependencies between address ranges
- Quality of optimization result depends on
  - Profiling information
  - Determinism
  - Periodicity

  } of application



Based on [6] Fig. 1

Content

- Introduction
  - Motivation
  - Background
- Reduction of Dynamic Energy Consumption
  - Optimization Concept
  - Workflow
- Reduction of Static Energy Consumption
  - Optimization Concept
  - Workflow and Example
- Conclusion

Optimization Concept

Identification of an optimal heterogeneous memory architecture (allocation) and application binding based on profiling information of a periodic application.

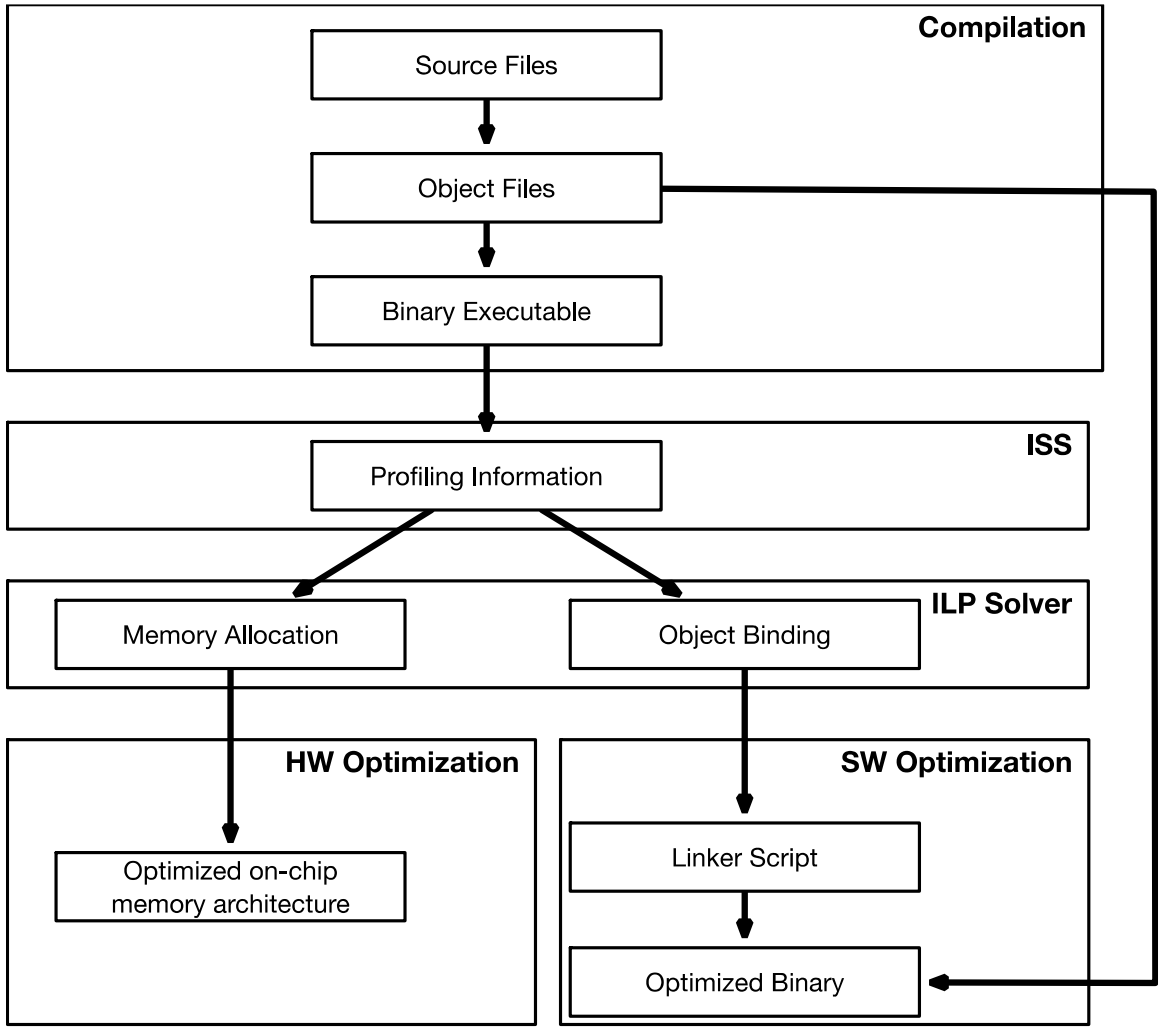
$$E_{mem} = E_{dyn} + E_{stat}$$

E_{dyn}	Dynamic energy consumption from read/write accesses
$E_{stat} = P_{leak} * T$	Static energy consumption due to leakage (period T)

- We proposed a **mathematical model** to perform the **combined optimization** of memory instance **allocation and binding** of memory segments to instances [5]
  - Minimizes total dynamic energy consumption over runtime
  - Integer Linear Programming (ILP) ensures global optimum

Optimization Workflow

- Considers α and β
- Experiments
  - Up to 8 memories
  - IBM CPLEX ILP solver
- Results
  - PPC405 platform
  - Energy savings up to 82%
  - Application-specific



Based on [5] Fig.3

Content

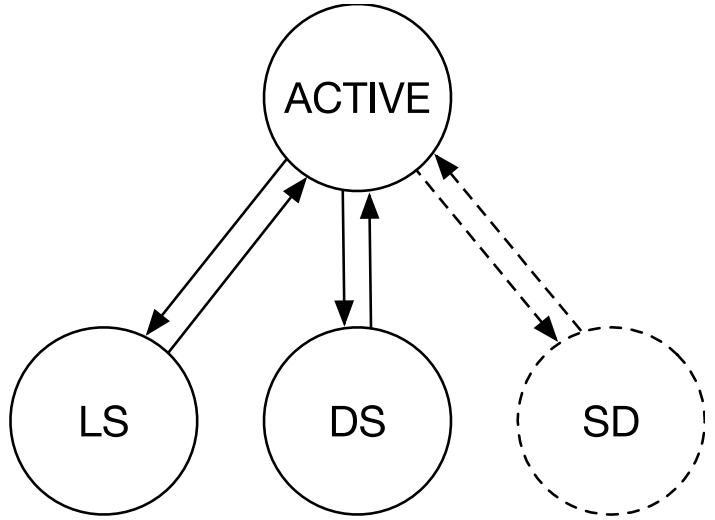
- Introduction
  - Motivation
  - Background
- Reduction of Dynamic Energy Consumption
  - Optimization Concept
  - Workflow
- Reduction of Static Energy Consumption
  - Optimization Concept
  - Workflow and Example
- Conclusion

Modeling of Low-power Modes

- Advances in technology scaling lead to increasing relevance of leakage
- Expected to exceed 50% of the total system power consumption [2]

- Adding power states to the memory subsystem model
- At each point in time, a memory is in
  - State $\{ACT, LS, DS, SD\}$
  - Transition $\{ACT \rightarrow LS, LS \rightarrow ACT, \dots\}$
- Example (two memories)

  $C = \{(ACT, ACT), (ACT, LS), \dots\}$



Optimization Concept

Identification of a heterogeneous memory architecture (allocation), application binding, and low-power mode scheduling based on profiling information of a periodic application.

$$E_{mem} = E_{dyn} + \underbrace{E_{leak} + E_{mode}}_{E_{stat}}$$

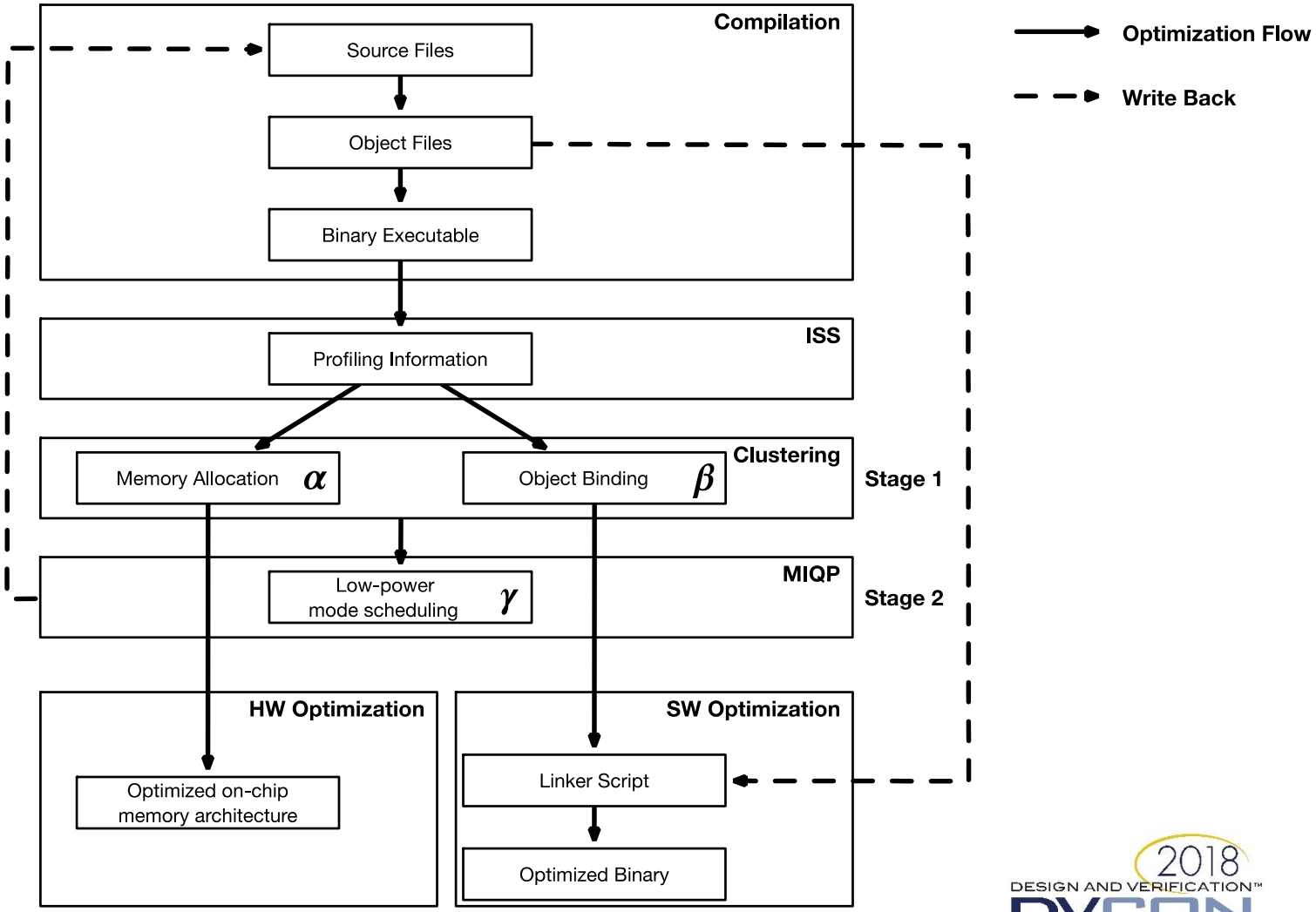
E_{leak} Energy consumption due to leakage (depending on low-power mode)

E_{mode} Energy penalty due to low-power mode changes

- Low-power modes allow the consideration of peak power constraints!

Optimization Workflow

- Supported ISS Alternatives
  - ARMv6-M ISS [6]
  - SystemC ARMv6-M VP (TLM) (www.soclib.fr)
  - Multicore VP (cooperation with RWTH Aachen)
- Clustering Alternatives
  - Dynamic energy optimization (see above)
  - Graph partitioning
  - Min-cut clustering
  - Modularity clustering
- MIQP Solver
  - Gurobi



Example

- Experimental result for the blowfish benchmark from the MiBench Suite [7]

Preparation

- Build application
- Run ISS (profiling)
- Results (JSON format)


```
{
 "profileId": 12,
 "profileType": "c",
 "profileName": "main",
 "startAddr": "0x00000e24",
 "endAddr": "0x00000f08",
 "size": 228,
 "dutyCycle": 7710544,
 "numReads": 6751633,
 "numWrites": 0,
 "depsVector": [ "p1": 0, "p2": 0,
                 "p3": 0, "p4": 0,
                 "p5": 7796, ... ]
} ...
```


Stage 1

- Determine α and β
- 32nm node memories
- Modularity Clustering
  - 4 memories (cluster)
  - Generate linker script

```
MEMORY
{
   m1 (rwx) : ORIGIN = ..., LENGTH = 8K
   m2 (rwx) : ORIGIN = ..., LENGTH = 8K
   m3 (rwx) : ORIGIN = ..., LENGTH = 512K
   m4 (rwx) : ORIGIN = ..., LENGTH = 8K
} ...
```

- Reduction to 27.54%

Stage 2

- Determine γ
- Solve MIQP
- Configurations

BF_cfb64_encrypt	DADD
BF_encrypt	DADD
BF_set_key	DADD
Reset_Handler	DAAD
exit	DAAD
main	**DAAD**
memcpy	DADA
putchar	DAAD
...	

- Reduction to 26.66%

Content

- Introduction
  - Motivation
  - Background
- Reduction of Dynamic Energy Consumption
  - Optimization Concept
  - Workflow
- Reduction of Static Energy Consumption
  - Optimization Concept
  - Workflow and Example
- Conclusion

Conclusion

- Due to the large share of system power consumption that can be attributed to the memory subsystem it is a promising target for optimizations.
- Profiling-based optimization of the memory subsystem allows savings of over **80%**.
- Largest gain comes from the optimization of dynamic energy through application-specific **allocation and binding** of memory segments to instances.
- Nevertheless, static power optimization through modeling of low-power modes becomes increasingly important with **technology scaling**. Even more because it enables the definition of **peak power** constraints.

References

[1] S. P. Mohanty, N. Ranganathan, and S. K. Chappidi. 2006. ILP Models for Simultaneous Energy and Transient Power Minimization During Behavioral Synthesis. ACM Transactions on Design Automation of Electronic Systems 11, 1 (2006), 186–212. https://doi.org/10.1145/1124713.1124725

[2] F. Menichelli and M. Olivieri. 2009. Static Minimization of Total Energy Consumption in Memory Subsystem for Scratchpad- Based Systems-on-Chips. IEEE Transactions on Very Large Scale Integration (VLSI) Systems 17, 2 (2009), 161–171. https://doi.org/10.1109/TVLSI.2008.2001940

[3] A. Mathur and L. Minwell. 2009. Memory Power Reduction in SoC Design Using PowerPro MG. (2009). https://www.design-reuse.com/articles/21806/memory-power-reduction-soc-design.html Last visited on 12/05/2017.

[4] M. Loghi, O. Golubeva, E. Macii, and M. Poncino. 2010. Architectural Leakage Power Minimization of Scratchpad Memories by Application-Driven Subbanking. IEEE Trans. Comput. 59, 7 (2010), 891–904. https://doi.org/10.1109/TC. 2010.43

[5] M. Strobel, M. Eggenberger, and M. Radetzki. 2016. Low power memory allocation and mapping for area-constrained systems- on-chips. EURASIP Journal on Embedded Systems 2017, 1 (2016). https://doi.org/10.1186/s13639-016-0039-5

[6] M. Strobel and M. Radetzki. 2017. Hybrid instruction set simulation for fast and accurate memory access profiling. In Proc. of the 13th Workshop on Intelligent Solutions in Embedded Systems (WISES). 23–28. https://doi.org/10.1109/WISES.2017.7986927

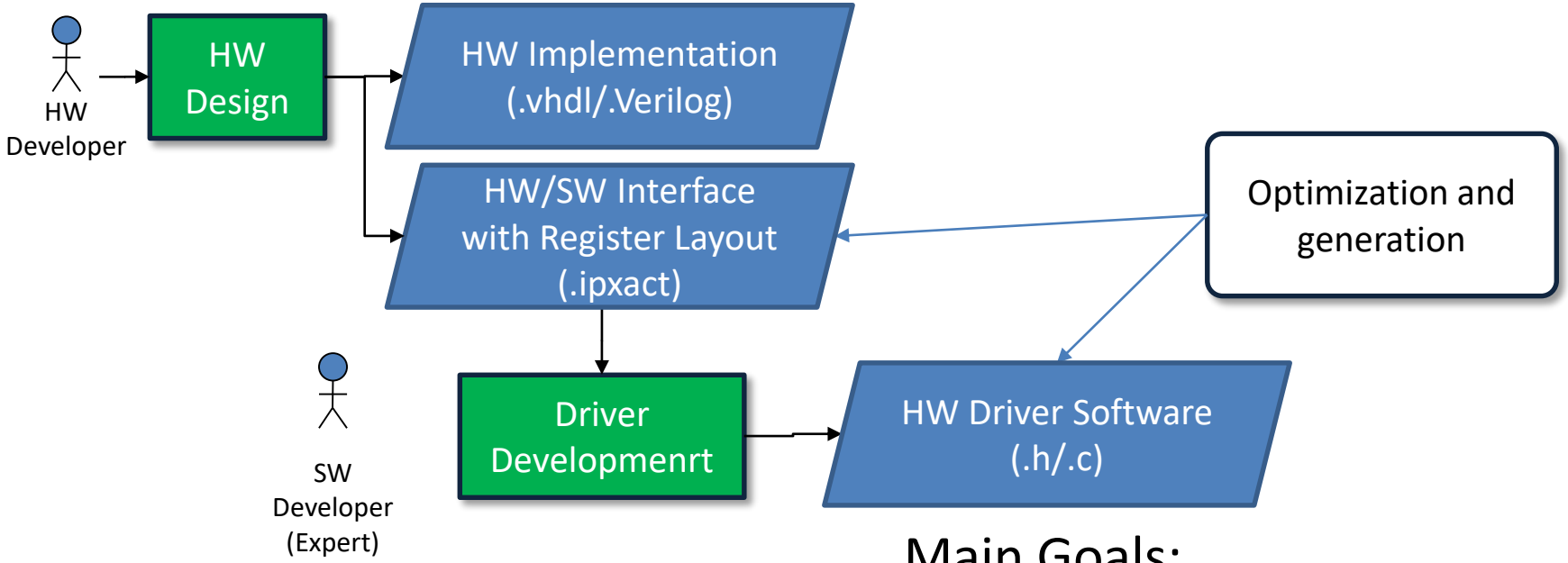
[7] M. R. Guthaus, J. S. Ringenberg, D. Ernst, T. M. Austin, T. Mudge, and R. B. Brown. 2001. MiBench: A free, commercially representative embedded benchmark suite. In Proc. of the 2001 IEEE International Workshop on Workload Characterization. IEEE. https://doi.org/10.1109/WWC.2001.990739

[8] SL Coumeri, DE Thomas, in *Proc. International Symposium on Low Power Electronics and Design (ISLPED)*. Memory modeling for system synthesis (IEEE, Montery, CA, USA, 1998), pp. 179–184

Automatic HW-SW-Interface Generation and Optimization

Daniel Mueller-Gritschneder (TU Munich)

Motivation

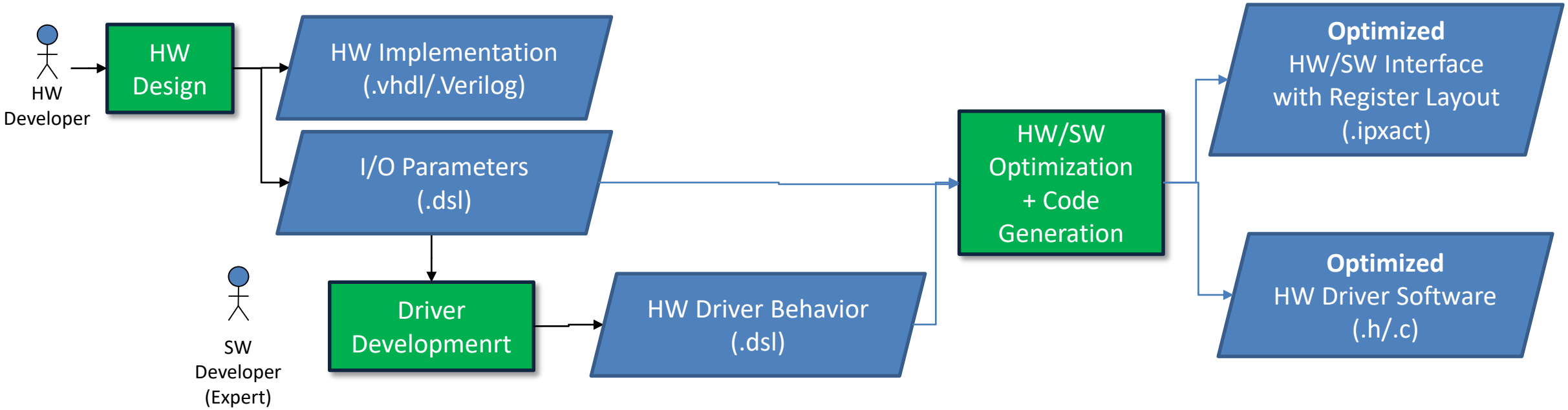


- High costs for driver development
- Expert knowledge for optimized memory required

Main Goals:

- Automatic driver generation from abstract description
- Optimized register bit-field layout in MCU
- Low memory footprint of drivers by generation & optimization

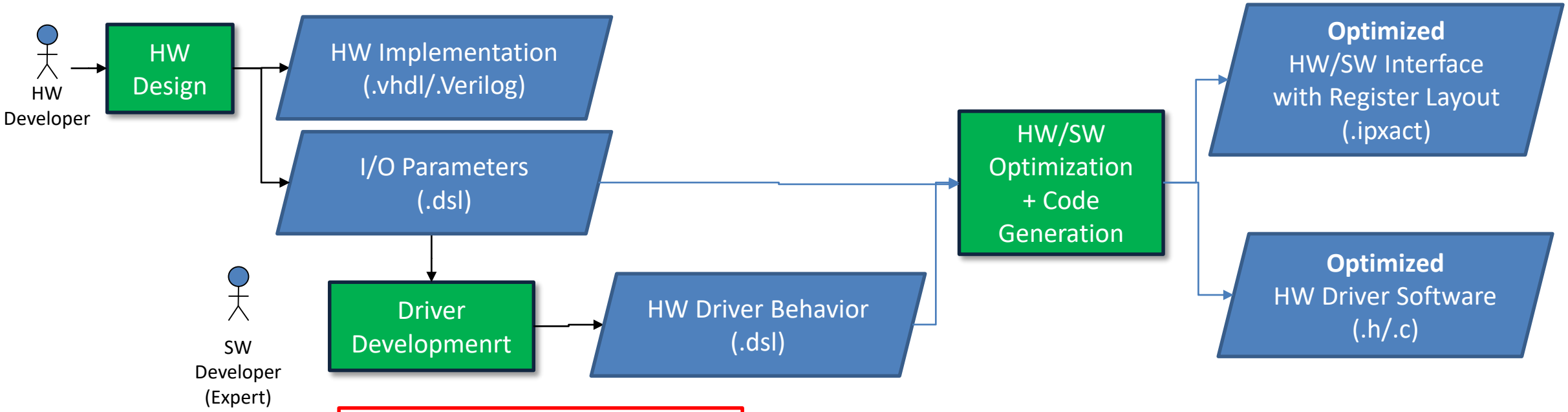
Optimization with Driver Model



How to specify HW I/O parameters and Driver behavior?

How to optimize the HW/SW interface?

Optimization with Driver Model



How to specify HW I/O parameters and Driver behavior?

How to optimize the HW/SW interface?

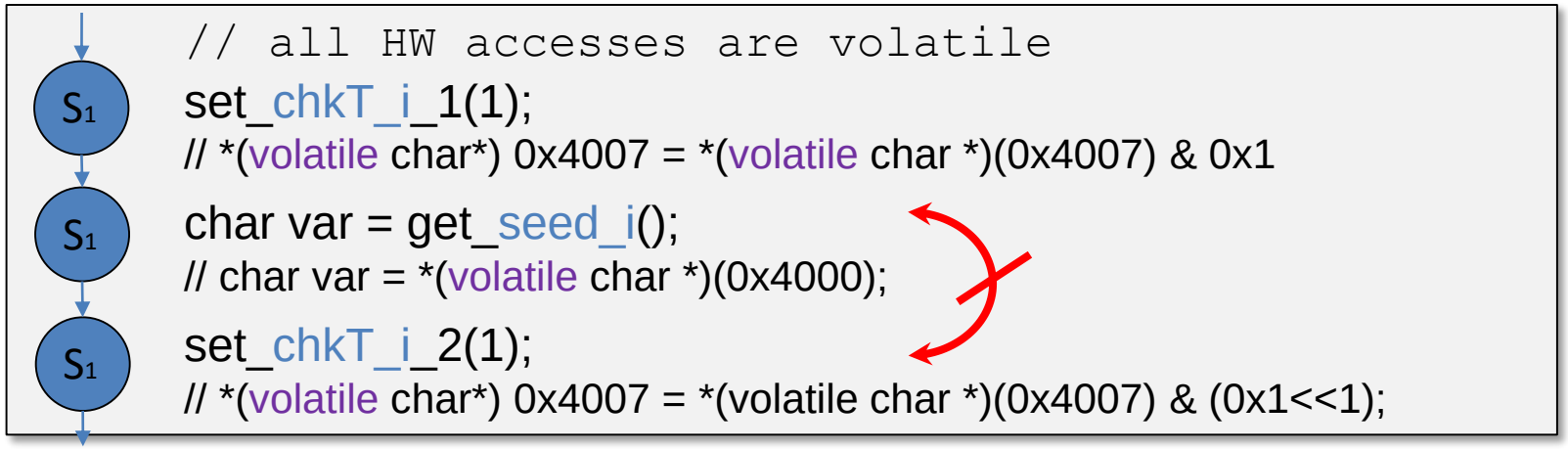
Driver Models and DSLs

- Devil[1], NDL[2], HAIL[3], Laddie[4]:
  - Unintuitive to C programmers
  - Fixed register bit-field layout
  - More restricting description for the side effects
  - Not focused on optimization of HW/SW interface

- [1] L. Reveillere and G. Muller, "Improving driver robustness: an evaluation of the Devil approach," 2001 International Conference on Dependable Systems and Networks, Goteborg, Sweden, 2001, pp. 131-140.
- [2] Christopher L. Conway and Stephen A. Edwards. NDL: A Domain-Specific Language for Device Drivers. In the proceedings of the ACM Conference on Languages, Compilers, and Tools for Embedded Systems (LCTES), Washington, DC, June 11-13, 2004
- [3] Sun, Jun, et al. "HAIL: a language for easy and correct device access." Proceedings of the 5th ACM international conference on Embedded software. ACM, 2005.
- [4] Wittie, Lea. "Laddie: The Language for Automated Device Drivers (Ver 1) Bucknell Computer Science Technical Report# 08-2." (2008).

Why not C for driver behavior?

- C-specifics:
  - Fix layout for registers and bitfields disallows to modify HW/SW interface
  - No bit-accurate types
  - Endianness fixed by processor
  - volatile keyword forbids most code optimizations (abstract automata of C model)



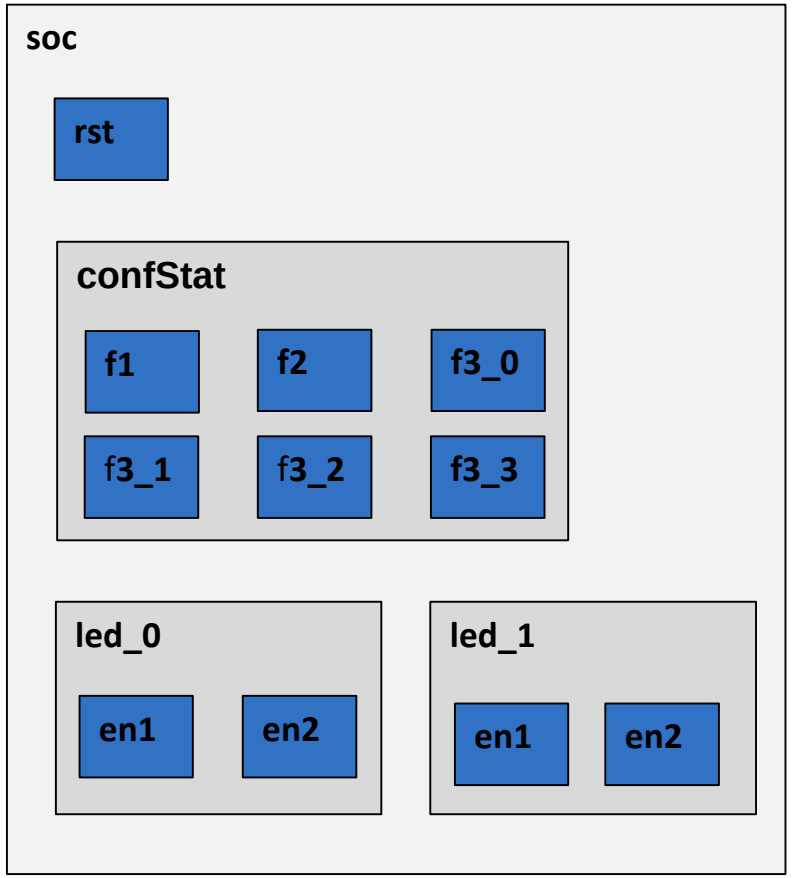
Own Driver Meta Model & DSL

- **Based on C-Syntax with some extensions**
- **Definition of HW I/O Parameters (bfGroup)**
  - Similar to C-struct, but less restricted
  - No fixed register bit-field layout
  - Bit-accurate types with array support
  - Detailed definition of HW side effects
  - Hierarchical organization for reuse
- **Definition of SW Parameters (swGroup)**
  - Without memory layout
  - C-types + bit accurate types + endianness
  - Hierarchically
- **Interface FSM:** Permitted API function calls in different states

DSL Example: Definition of HW I/O Parameters

```
bfGroup ledType{
  bit en1;
  bit en2;
};
bfGroup confStatType {
  uint3 f1;
  uint4 f2;
  uint3 f3[4];
};
bfGroup SoCType{
  bfGroup confStatType confStat;
  bfGroup ledType led[2];
  uint1 wse(confStat)
        wse(led) rst;
};
make_device(SoCType, soc);
```

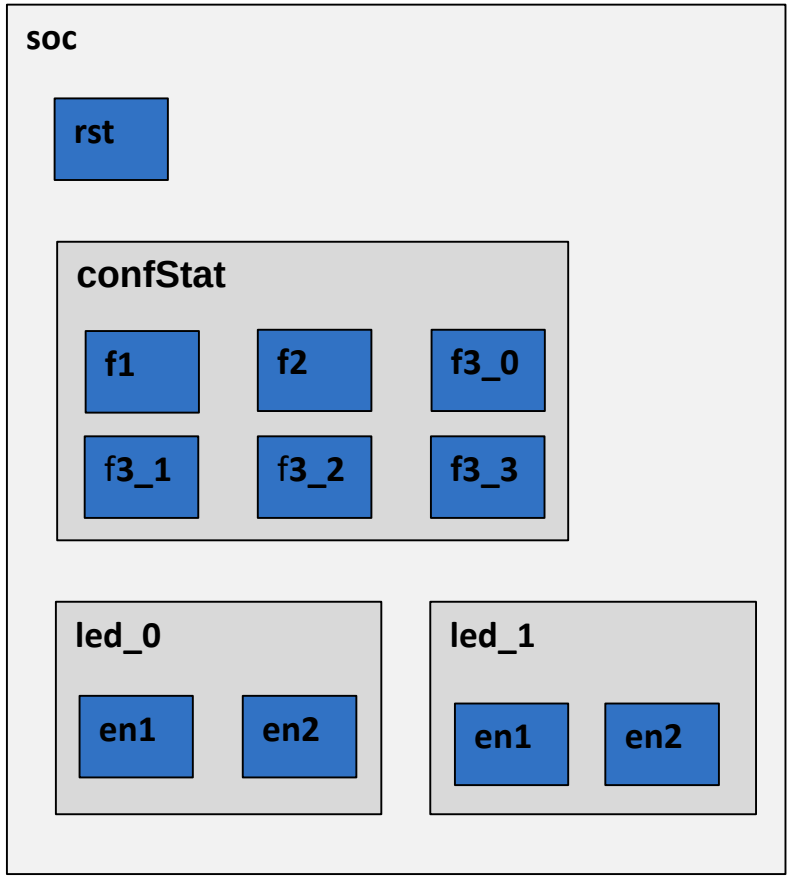
Write side effect



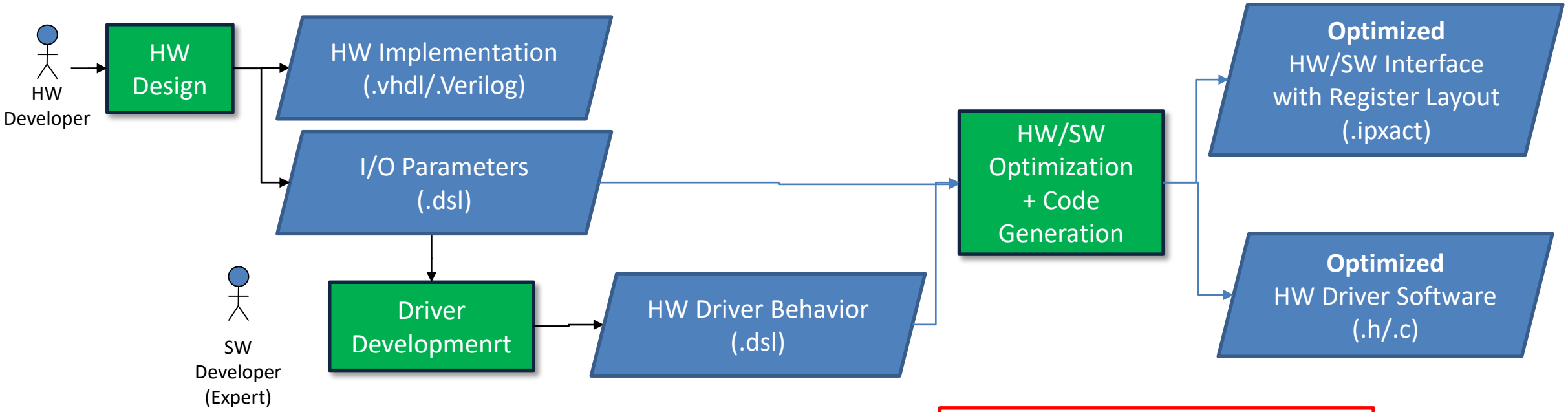
DSL Example: Driver Behavior Specification

```
void program_device()
{
  soc->confStat.f1 = 0;
  for(int i=0;i<4;i++)
  {
    soc->confStat.f3[i]=i;
  }
  soc->led[0].en2 = 1;
}

void blink_leds(
    bfGroup ledType* led)
{
  led->en2 = 1;
  led->en1 = 0;
}
```



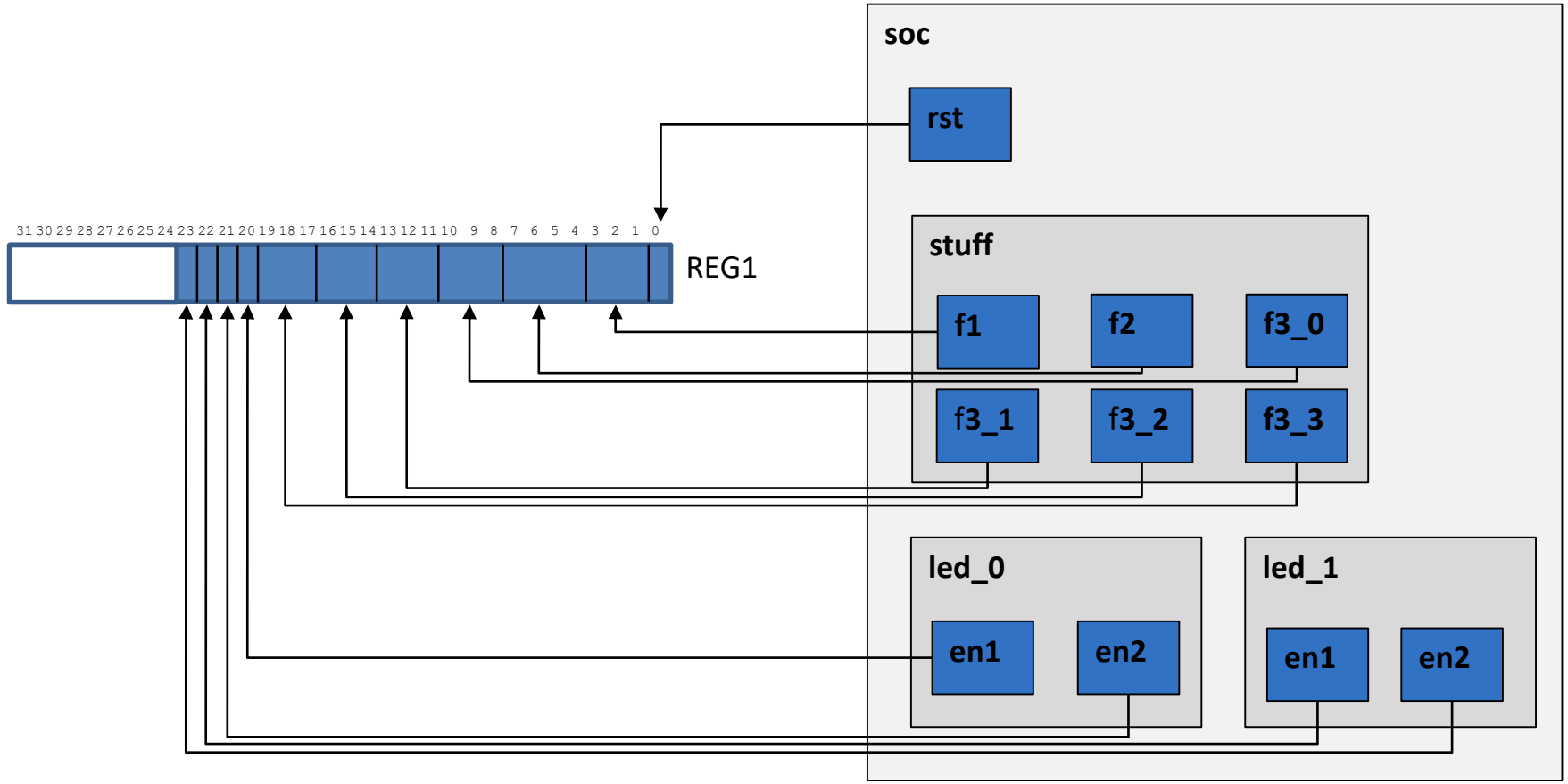
Optimization with Driver Model



How to specify HW I/O parameters and Driver behavior?

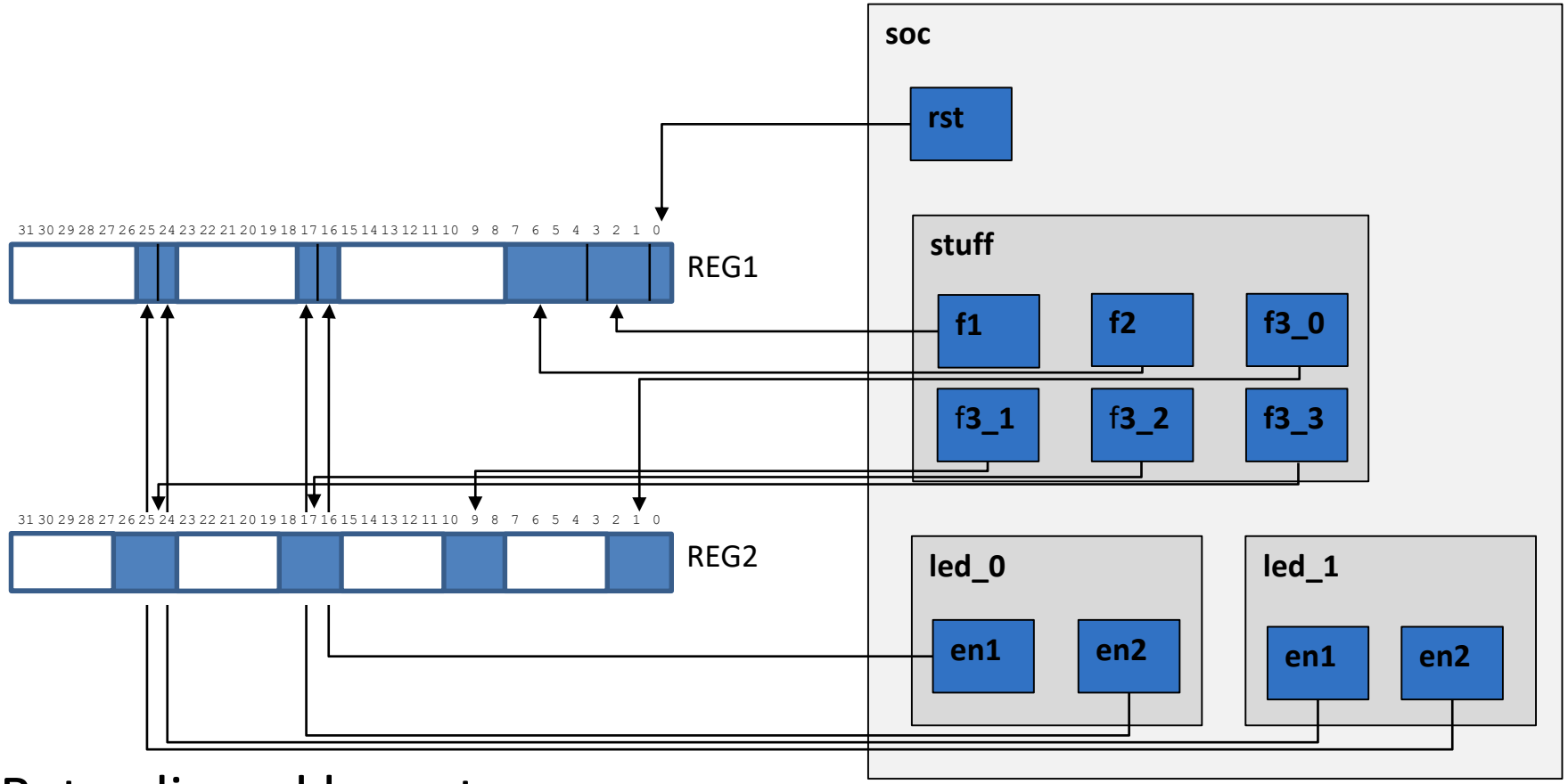
How to optimize the HW/SW interface?

Optimization of the HW/SW Interface



- Save number of Registers (saves MUXs)

Optimization of the HW/SW Interface



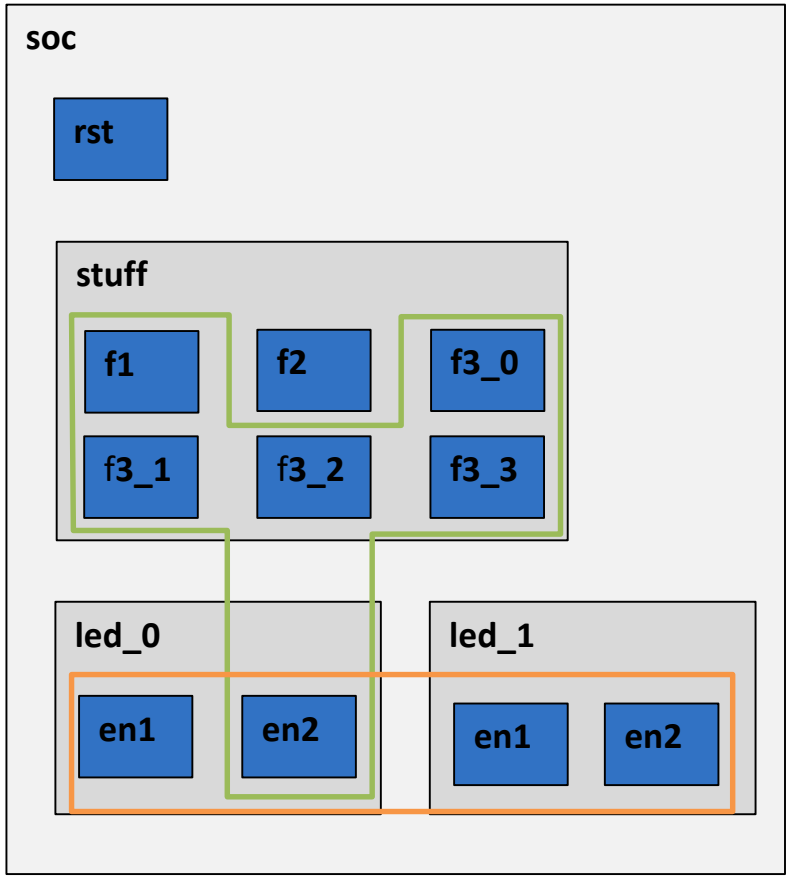
- Byte aligned layout
- Byte access here without shifts possible

Static Code Analysis -> Access Regions

- Within regions accesses can be combined
- Regions will be divided by data dependencies
- Regions have access frequencies

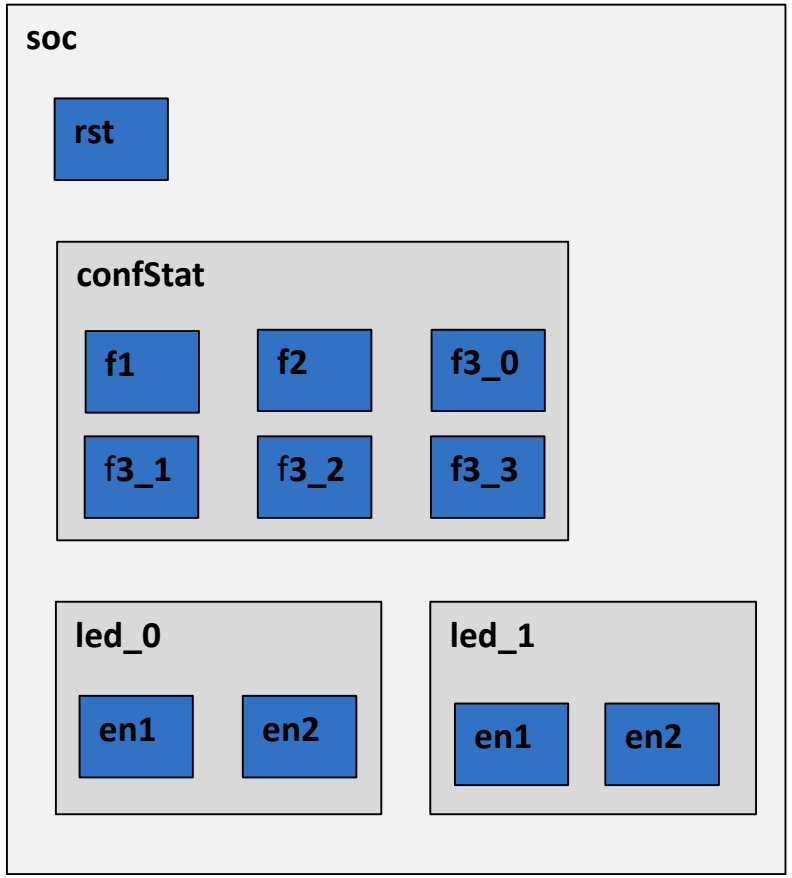
```
void program_device()
{
  soc->confStat.f1 = 0;
  for(int i=0;i<4;i++)
  {
    soc->confStat.f3[i]=i;
  }
  soc->led[0].en2 = 1;
}

void blink_leds(
     bfGroup ledType* led)
{
  led->en2 = 1;
  led->en1 = 0;
}
```



Optimized Generation of the Driver Code

```
void program_device()
{
  set_soc_confStat_f1(0);
  for(int i=0;i<4;i++)
  {
    set_soc_confStat_f3_i(i,i);
  }
  set_soc_confStat_f2(3);
  set_soc_led_0_en2(1);
}
```



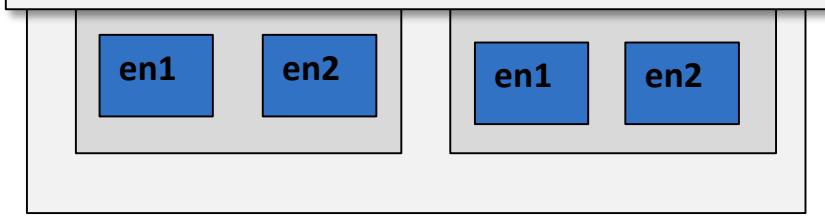
- Possible combination of consecutive HW accesses on the same register
- Map often consecutive used bit-fields to one register

Optimized Generation of the Driver Code

```
void program_device()
{
  set_soc_confStat_f1(0);
  for(int i=0;i<4;i++)
  {
    set_soc_confStat_f3_i(i,i);
  }
  set_soc_confStat_f2(3);
  set_soc_led_0_en2(1);
}
```

```
void program_device()
{
  set_soc_confStat_f1_f2(0,3);
  set_soc_confStat_f3(0,1,2,3);


  set_soc_led_0_en2(1);
}
```



- Possible combination of consecutive HW accesses on the same register
- Map often consecutive used bitfields to one register

Ongoing Work

- Evaluation of costs for different HW/SW Interfaces and generated Driver Codes
- Implementation of optimization heuristic
- Apply method to optimize a Pulpino RISC-V SoC

	Single Bit-field Register	Shared Bit-field Register
Read single BF	C_{io}	$C_{io} + 2\ C_{instr}$
Write single BF	C_{io}	$2\ C_{io} + 3\ C_{instr}$
Read whole Register	C_{io}	$C_{io} + (2n-1)\ C_{instr}$
Write whole Register	C_{io}	$C_{io} + (2n-2)\ C_{instr}$
Read n BFs	$n\ C_{io}$	$C_{io} + 2n\ C_{instr}$
Write n BFs	$n\ C_{io}$	$2\ C_{io} + (2n+1)\ C_{instr}$

Conclusions

- Our driver DSL supports:
  - Compact and hierarchical description of I/O parameters and driver behavior
  - optimized generation of the register layout
  - to minimize hardware access number of firmware by combining accesses

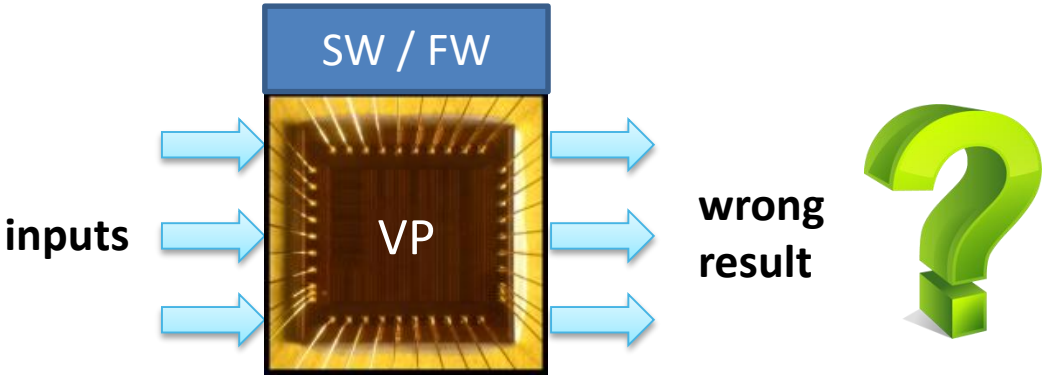
- DSL avoids
  - the strict volatile accesses in C, but considering side effects
  - a fixed bitfield layout (and the effort of developing it)

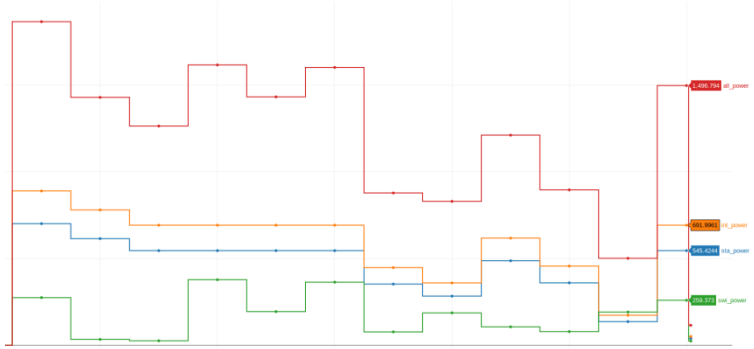
RISC-V based Virtual Prototype for Efficient Simulation of Firmware-based Designs

Vladimir Herdt (Univ. of Bremen)

Early Simulation of Firmware-based Designs

- Detect errors early in the design flow
- Enables early SW/FW development
- Functional Verification
- Non-Functional Verification
  - Performance / Timing
  - Energy Consumption
- SystemC-based Virtual Prototypes





www.systemc.org

RISC-V (1)

- Completely open ISA that is freely available
- No license costs involved
- Efficient and versatile design
- High-performance to small embedded devices
- Widely adopted

RISC-V (2)

- Mandatory Integer Instruction Set “I” (~47 instrs.)
  - 32/64/128 Bit
- + Optional Extensions
  - “M”, “A”, “F”, “D”, etc.

RV32IMAFD = RV32G

- Control and Status Registers (CSRs) and Environment Interaction

VP Overview (1)

- RV32IM(A) + Machine Mode CSRs
- Implemented in SystemC/C++
  - TLM-2.0 compliant
  - approx. 3000 LOC (w/o comments, blanks)

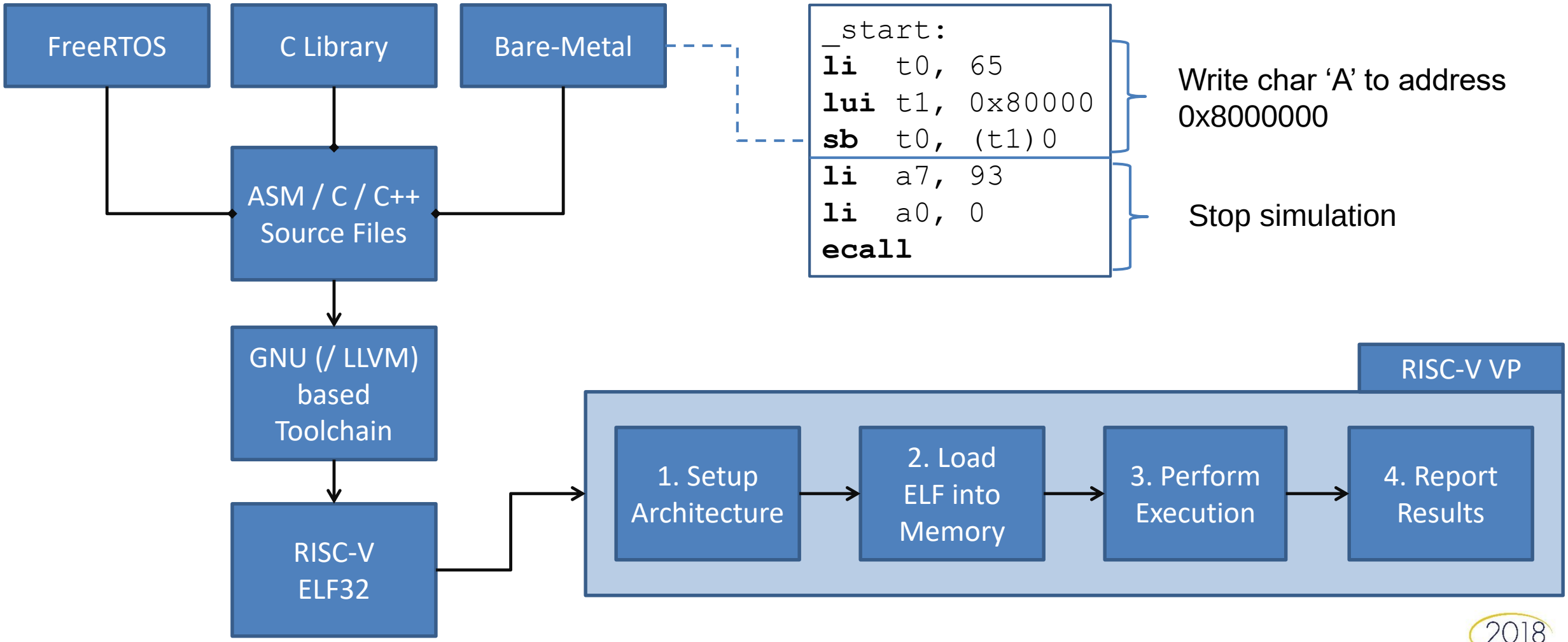
- **Open Source**
  - https://github.com/agra-uni-bremen/riscv-vp
  - MIT license
- Overview paper at FDL 2018
  - http://www.informatik.uni-bremen.de/agra/doc/konf/2018FDL_RISCV_VP.pdf

VP Overview (2)

- **Components:**
  - Core, Bus, Memory,
  - Interrupt Controller,
  - Peripherals (Sensor, Timer, DMA, Terminal)

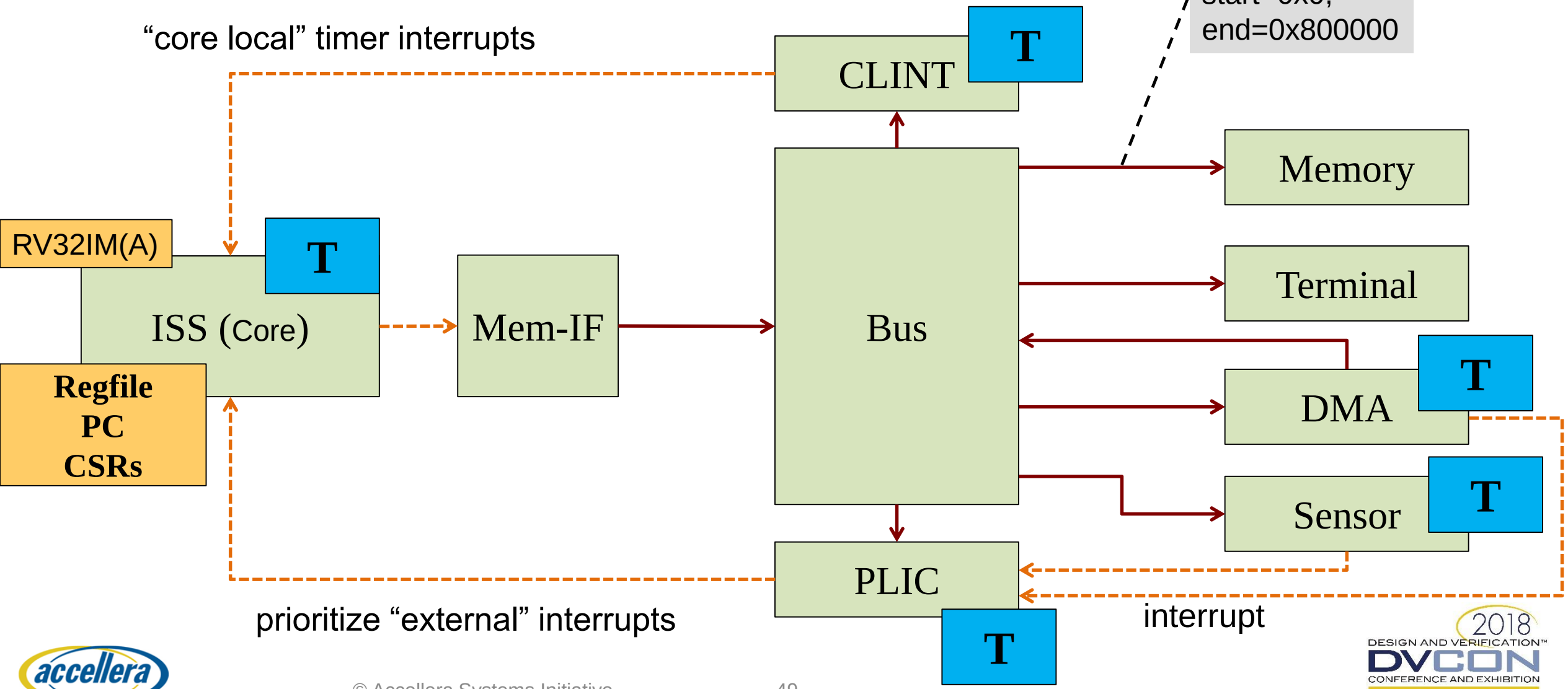
- **Supports:**
  - Interrupts, Syscalls, CLIB, GCOV
  - **Recently:** GDB, FreeRTOS, FAT, basic ethernet

VP-based Simulation for RISC-V



configurable / extendable

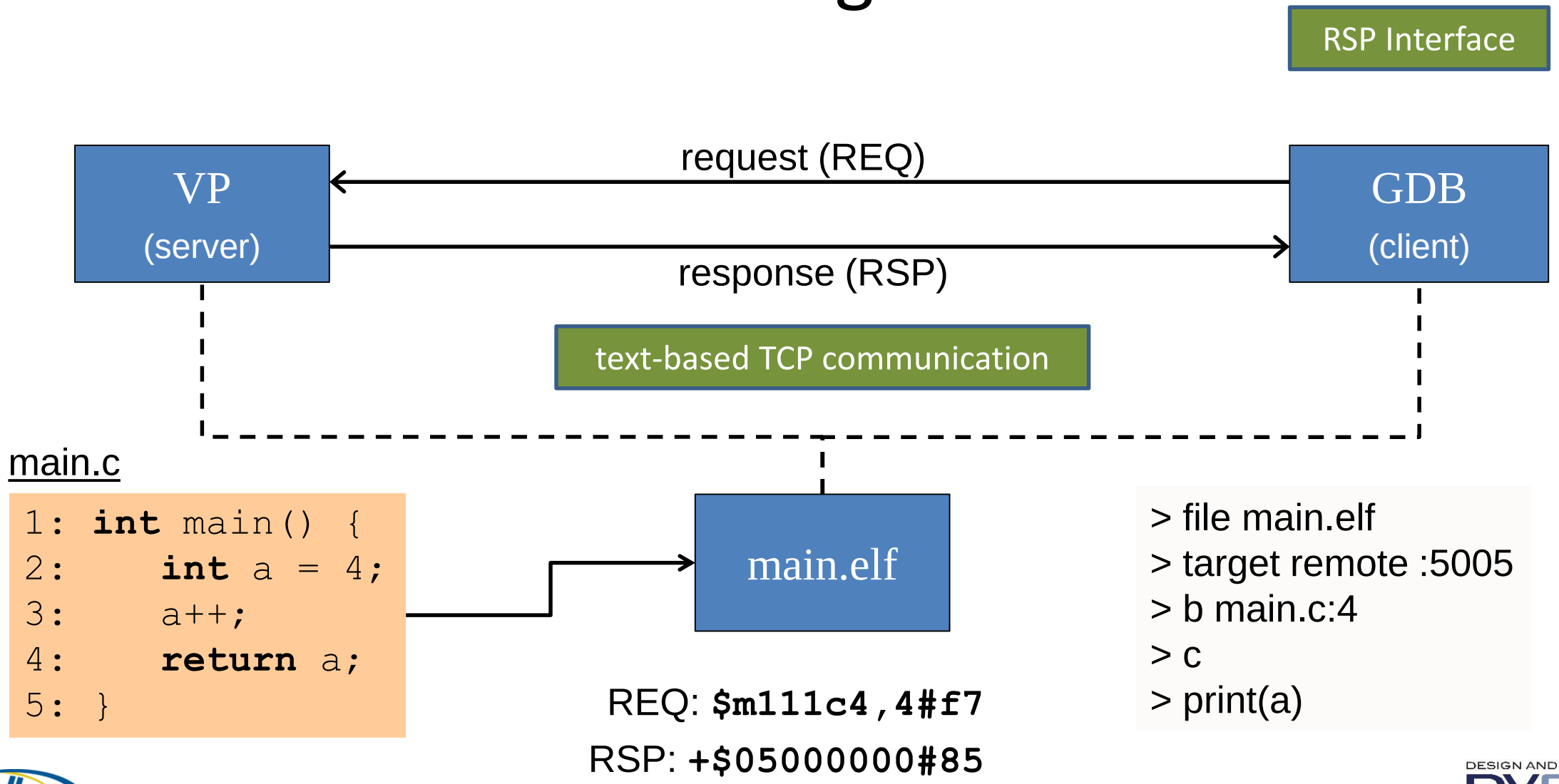
VP Architecture



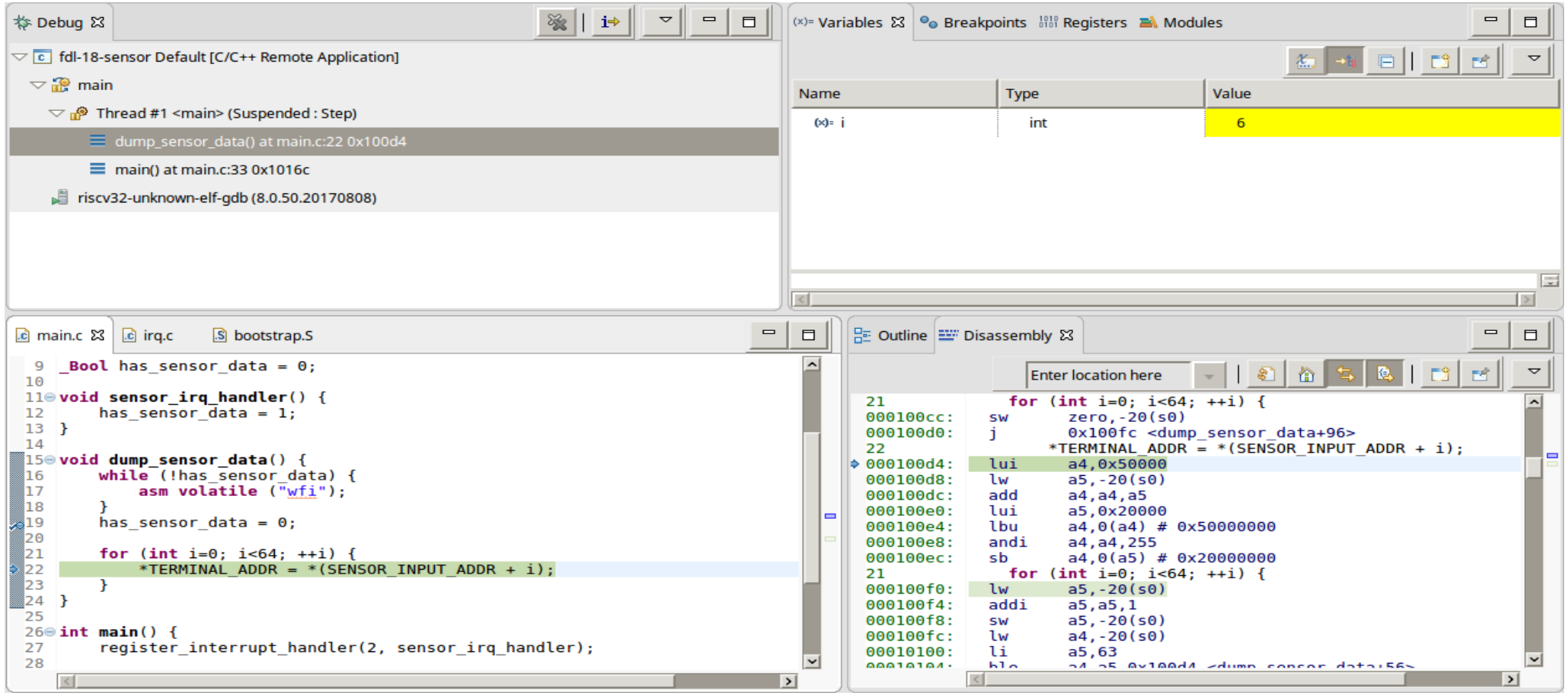
Timing Model

- Simple Instruction Accurate
  - Fixed execution times for each instruction
  - Easy to configure
- TLM blocking transactions
  - b_transport(*tlm::generic_payload* &payload, *sc_core::sc_time* &delay)
  - Peripherals increment the delay parameter
- More precise models can be integrated

GDB Integration



Example: FreeRTOS + Eclipse GDB



https://github.com/agra-uni-bremen/riscv-freertos

Performance Optimization

- DMI for main memory access
  (core concept: char* access to memory)
  - DMI for instruction fetching
  - DMI for data access

- Temporal decoupling in CPU core
  - Evaluate different local time quantums

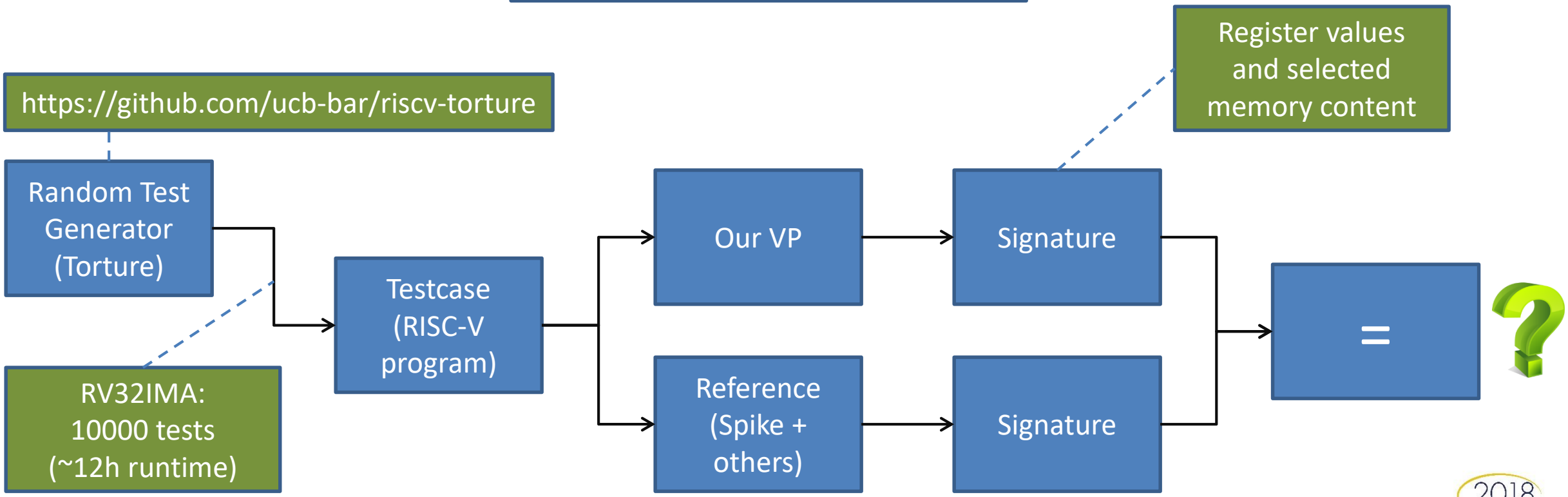
Performance Evaluation

- Simulation time in seconds (T.O. set to 4 hours)
- VP ~15-20 million instructions per second (AMD 2.8GHz)

Benchmark	RTL Sim	VP Sim	+i_dmi	+d_dmi	+q10	+q100
mergesort/s	56.70	0.39	0.35	0.35	0.32	0.29
primes/s	823.11	1.73	1.01	0.96	0.59	0.49
qsort/s	64.50	0.40	0.35	0.34	0.31	0.30
sha512/s	1307.23	3.23	1.87	1.57	0.90	0.71
mergesort	T.O.	197.32	107.89	86.48	41.17	27.77
primes	T.O.	2400.32	1214.71	1089.36	542.46	387.09
qsort	T.O.	1204.98	698.50	510.70	262.93	116.73
sha512	T.O.	2773.60	1556.02	1302.75	616.10	432.52

VP (ISS) Testing

- RISC-V ISA tests from Berkeley: passed (57 tests, RV32IMA) https://github.com/riscv/riscv-tests



Conclusions

- RISC-V based Virtual Prototype for efficient simulation of firmware-based designs
- Future work:
  - 64 Bit Support and ISA Extensions (e.g. compressed instructions)
  - Verification
    - RISC-V VP Model:
      Symbolic simulation using SISSI [1]
      UVM / CRAVE [2]
    - SW running on RISC-V VP:
      Symbolic execution to check user assertions
  - Enhanced performance and power estimation
- Our open source RISC-V projects: www.systemc-verification.org/riscv-vp

[1] Verifying SystemC using Intermediate Verification Language and Stateful Symbolic Simulation (TCAD 2018)
[2] CRAVE: An Advanced Constrained RAndom Verification Environment for SystemC (SoC 2012)

Properties-First Design: A New Design Methodology for SoC Hardware and Low-Level Software

Tobias Ludwig (Univ. of Kaiserslautern)

accellera SYSTEMS INITIATIVE

2018 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION EUROPE

Status of Formal Verification

The biggest hurdle

black box verification

(simulation)

white box verification

(property checking)

Property Checking:
More like *design* than like *verification* !

"Stimulus, response! Stimulus, response! Don't you ever think?"

Vision

Formal RTL verification should:

- do more than *bug hunting*
- help to emancipate ESL models from prototypes to golden design models
- support new abstraction principles between electronic system level models (ESL) and low-level implementations (RTL)

Abstraction from RTL to ESL

Establish a sound relationship between ESL and RTL:

- System-level behavior is described in terms of a PPA
- Each operation of the ESL is related to objects of the RTL by formal properties
- “Operation properties” are proven on the RTL
- If all properties hold then the RTL is a sound refinement of the ESL.

Soundness

„Soundness“:

Any property fulfilled by the ESL design is guaranteed to hold also on the RTL.

Theorem shown for LTL properties (in J. Urdahl, D. Stoffel, W. Kunz: "Pathredicate Abstraction for Sound System-Level Models of RT-Level Circuit Designs", *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, Vol. 33, No. 2, Feb. 2014, pp. 291-304.)

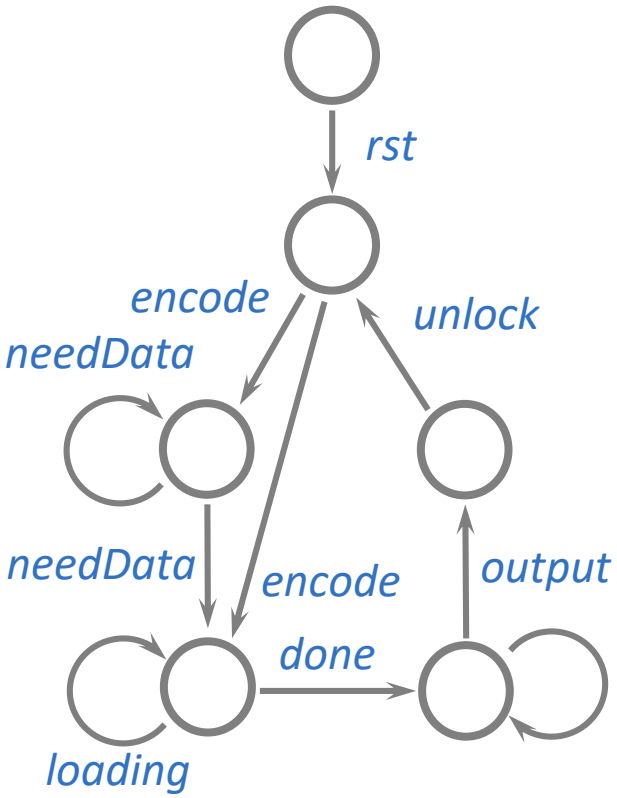
Soundness implies that:

- Verification results obtained at ESL translate to the RTL
- Global verification tasks can be moved from the RTL to the ESL
- Significantly less chip-level simulation is required

Property Driven Development

Basic Idea

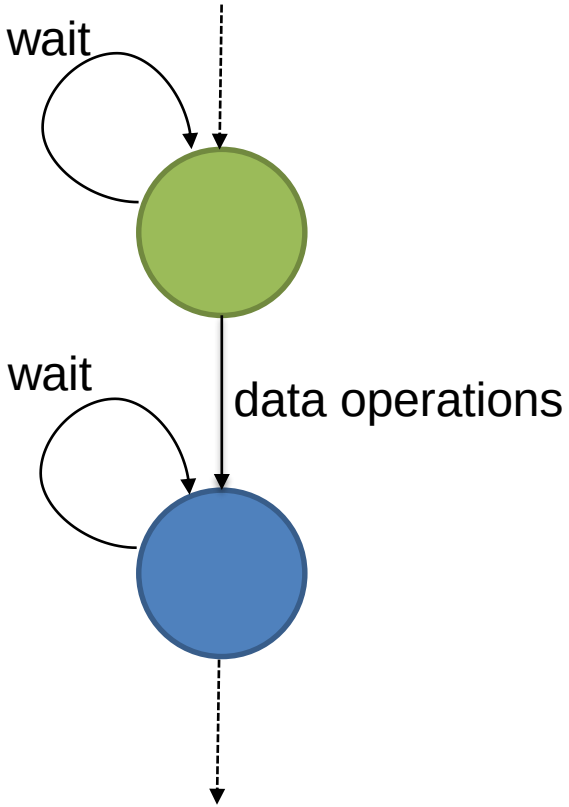
- System model as formal spec:
  - Designable/implementable subset of SystemC (*SystemC-PPA*)
  - Generate complete set of properties from system-level model
- Refine generated properties (templates) during design process
- Prove properties on final implementation



JPEG encoder

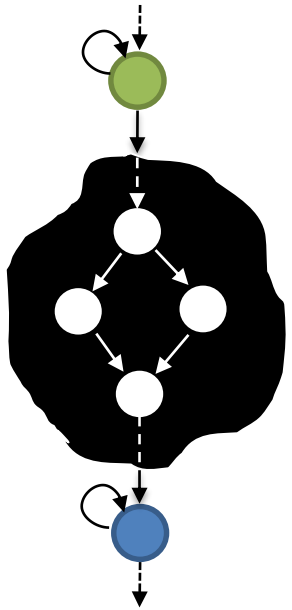
SystemC-PPA to PPA

```
SC_MODULE(example){
      SC_CTOR(example){ … }
      int data;
      void fsm(){
            //some code …
            port→read(data)
            //data operations
            port →write(data)
            //some code
      }
}
```



Operation properties

Operations start and end in important states and have finite length



n cycles

Important states subsume millions of concrete states

property *my_operation*;
assume:
at *t:* $a_{start}(S)$; // starting imp. state
at *t*: $a_0(X)$; // trigger…
prove:
at *t+n*: $c_1(X,Y)$;
at *t+n*: $c_n(X,Y)$;
at *t+n*: $c_{end}(S)$; // ending imp. state
endproperty;

S: state variables, *X*: inputs, *Y*: outputs

Benefit

- What is the ***promise?***
  - Correct-by-construction design refinements
  - Support for aggressive optimization techniques
  - Increased design productivity
  - Improved closure w.r.t. non-functional design goals
    - Power
    - Safety

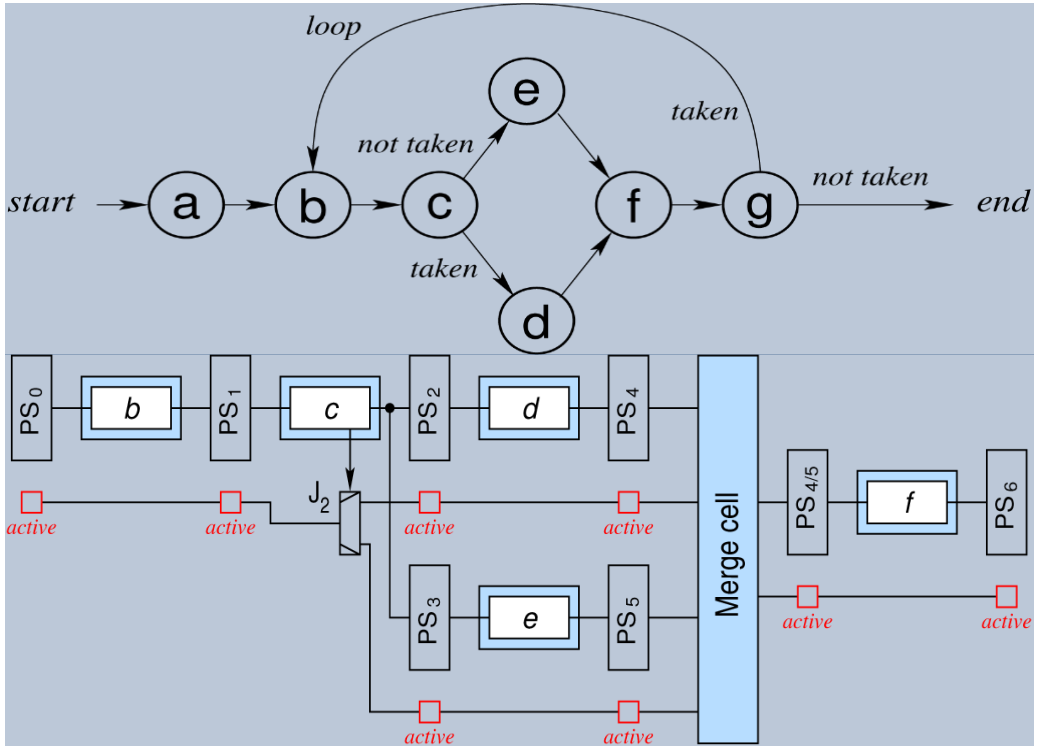
Program Netlist Model

Formal analysis is interleaved with unrolling

- Path pruning
- Path merging
- Unrolling loops

Program Netlist

- combinational circuit
- compactly represents all execution paths



HW-dependent software model

- PNs include explicit information for a given program on:
  - all possible execution paths (unlike traditional symbolic execution)
  - the address spaces reached by every instruction
  - all possible input/output access sequences to peripheral hardware components and to shared memory
  - all possible effects of the program on the program-visible hardware registers

Tooling

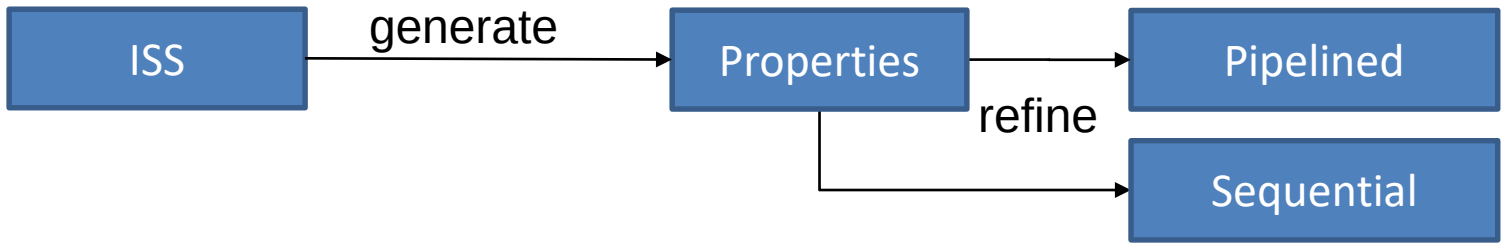
- The tool *SCAM* (“SystemC Abstract Model”):
  - Analyses a given SystemC model for compliance with the designable subset
  - Supports by refining the model into a SystemC-PPA
  - Automatically generates the properties
  - Manual explaining PDD from a practical point of view

Available on GitHub: github.com/ludwig247/SCAM

Contribution

- Instruction set simulator (ISS) as a designable SystemC-PPA
- Two RTL implementations, (1) sequential and (2) pipelined:
- Implementations are sound refinements of the same ESL

Available on GitHub: github.com/ludwig247/SCAM



Results ESL

- Work effort for implementation:
  - Sequential: 2 Weeks
  - Pipelined: 4 Weeks
- Work effort for refining the properties:
  - Sequential: 3 person days
  - Pipeliend: 6 person weeks

Results RTL

RTL simulation results

Program	ISS	RTL seq.	RTL pipl.
Prime numbers	5s	55s	83s
Fibonacci	1s	109s	135s
Bubble sort	8s	133s	208s

RTL implementation results

Design	Inp./Out.	FFs	LoC
Sequential	39/70	1881	1000
Pipelined	39/70	2500	2000

Industrial Case Study

SONET/SDH FRAMER — ORIGINAL DESIGN AND REDESIGN

	RTL Design			PPA		
Module	inp./out.	FFs	LoC	inp./out.	var.	states/ops.
Framer (or.)	549/280	4.2k-47k	27k	7/6	4	4/13
Monitor(or.)	20/6	30	850	3/1	2	2/9
Framer (re.)	549/280	3.9k-42k	12k	7/6	4	4/13
Monitor (re.)	20/6	425	92	3/1	2	2/9

Work effort: bottom-up: 2 PMs | top-down PDD: 1 PM

Conclusion

- **Property Driven Development (PDD):**
  - Enables a top-down hardware development
  - Results in a formally sound correct-by-construction design
  - No formal verification knowledge required
- In practice, PDD is based on:
  - The provided open-source tool SCAM
  - State-of-the-art property checking
  - Specific methods for refining the properties

→ Shifting global design and verifications to the ESL!

Tutorial Conclusion

- Application-specific adaptivity by firmware
- Four firmware levels
- Four solutions
  - Optimization of memory subsystem
  - HW-SW Interface Generation
  - RISC-V based VP Simulation
  - Properties-First Design