SIEMENS

Domain Specific Acceleration with High-Level Synthesis

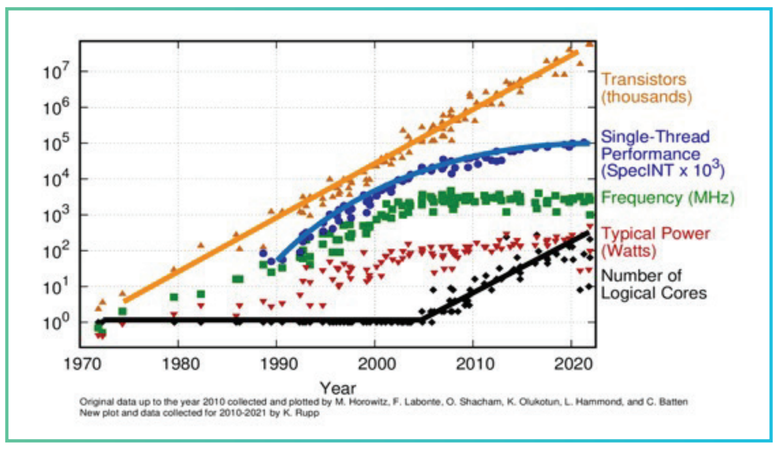
Siemens EDA { Petri Solanti, Russell Klein }

Background

- Software has many desirable characteristics
  - Inexpensive to create
  - Large developer community
  - Plentiful free open-source frameworks, tools, libraries, stacks, etc.
  - Easy to patch and update
    - Many systems can be updated over the air/network while in operation
    - Updates deliver bug fixes and new functionality

- But software is slow and inefficient
  - Programmer's model of serial, atomic instruction execution limits implementation parallelism
  - Large processors are exponentially power and area inefficient
  - Cannot simply use increasingly larger processors

- General purpose processors are reaching their limit
  - Moore's law continues
  - But single threaded performance for general-purpose compute has stalled
  - Adding cores is not effective as software is single threaded



Domain Specific Acceleration

- David Patterson advocates for "Domain Specific Architectures" to address this challenge

> "A New Golden Age for Computer Architecture: Processor Innovation to Enable Ubiquitous AI," a Keynote Presentation from David Patterson
>
> Edge AI and Vision Alliance Summit 2020

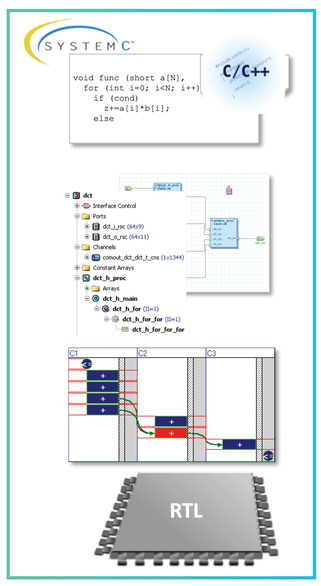
- Accelerators are needed to deliver both performance and efficiency gains Domain Specific Accelerators (DSA) extend the base ISA to optimize compute resources in a narrow domain

- Patterson proposes heterogenous compute based on RISC-V with ISA extensions, co-processors, and accelerators

- Creating a Domain Specific Accelerator
  - **Partition** – find compute bottlenecks in the base compute architecture by profiling complete applications, OS, and software stacks, identify opportunities for parallel computation
  - **Design** – the hardware implementation of the accelerator, create hardware and software interfaces
  - **Validate** – prove functionality, performance and efficiency of DSA as a standalone element, and integrate into the larger system of hardware and software

High-Level Synthesis

- Synthesis technology for transforming algorithmic C++ into synthesizable RTL suitable for AISC or FPGAs
  - Parses function source and creates control/dataflow graph
  - Walks graph to discover possible parallelism and data dependencies
  - Creates a set of state machines and data-path logic to implement the function
  - Writes synthesizable RTL description based on
    - user directives for loop unrolling and pipelining
    - target ASIC or FPGA technology
    - target implementation clock frequency



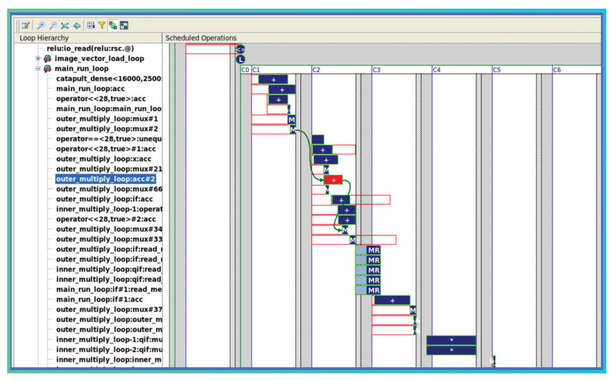
- HLS enables:
  - Faster design
  - Faster verification
  - Exploration of design alternatives

- HLS provides early insight into design characteristics
  - **Performance** – HLS schedules design, gives early latency and throughput
  - **Power and Area** – HLS gives pre-synthesis area estimates based on algorithm and target technology ASIC library and frequency

Power, Performance, and Area

- Early PPA estimates enable informed architecture decisions
  - PPA estimates will not be perfect, but are sufficient for comparison

- Throughput and latency are calculated from a clock cycle accurate schedule produced during synthesis



ASIC Library

Nand	Delay	Area	Dyn Power	Static Power
	0.145	0.023	1.1	0.3
	0.137	0.027	1.2	0.5
	0.121	0.034	1.4	0.7
	0.092	0.04	1.6	0.9
	0.071	0.045	2.3	1.1

- Area is estimated from a characterized library and based on combinatorial logic, fanout, and targeted F_{max}

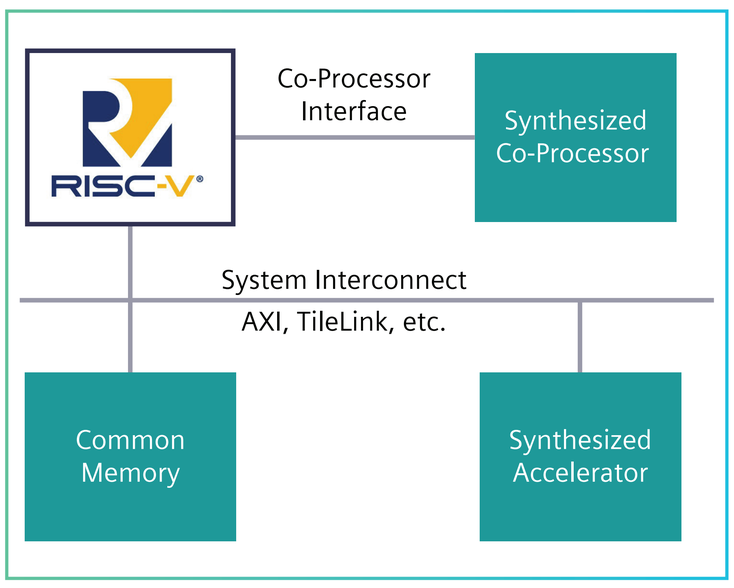
- Power is estimated from library data and switching activity derived from C++ testbenches

Integration with RISC-V Cores

- Accelerators can be deployed as co-processors through a co-processor interface or as bus based accelerators, connected to either local buses or system interconnects

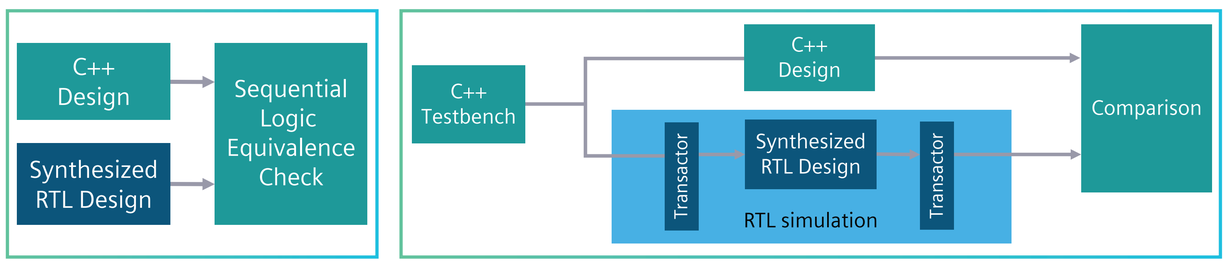
- Bus based accelerators can be designed as masters or slaves
  - HLS interface synthesis enables easy connection to popular bus protocols
  - Accelerators can signal interrupts on completion

- Software interacts with the accelerator through:
  - Co-processor instructions
  - Physically addressable control/status registers



Verification

- Verification is achieved through a combination of formal sequential logic equivalency checks and dynamic simulation using the C++ as a reference mode, with coverage for both C++ and RTL.



Example Results

- Experimental results for several compute intensive algorithms showing performance increase and energy saving per computation as compared to a RISC-V Rocket core on a 12 nm ASIC process.

Algorithm	Interface	SW perf/ HW perf	SW energy/ HW energy
SHA-256 hash	Co-Processor	25	49
Wakeword	Bus slave	213	980
Yolo Tiny	Bus master	1,123	1,480