ASIP University Day 2021

Application-Specific Processors (ASIPs) in System-on-Chip Design: Market, ASIP Designer Introduction & University Program

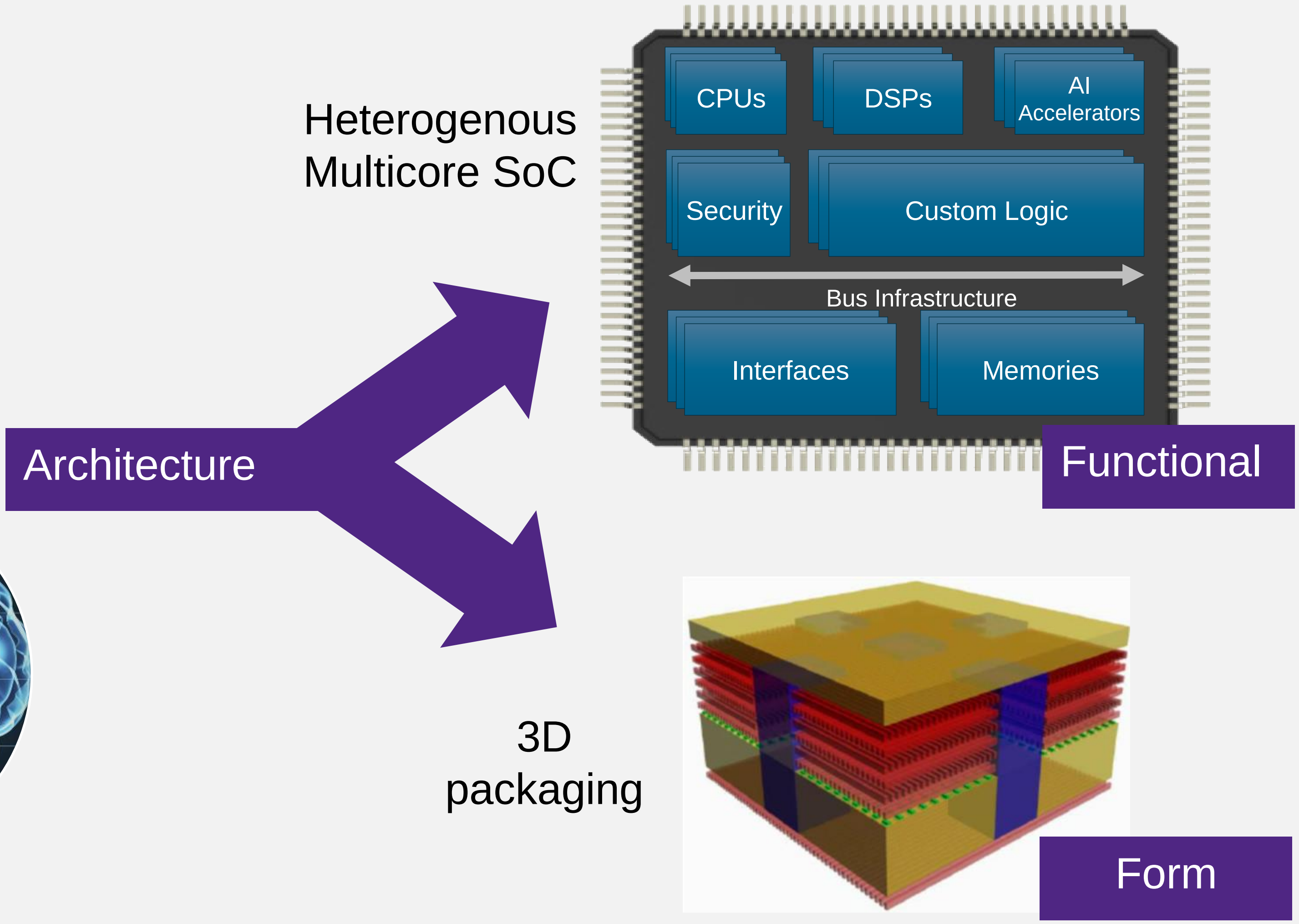
Patrick Verbist, Product Marketing Manager, ASIP Tools
Falco Munsche, Technical Marketing Manager, ASIP Tools

November 17, 2021

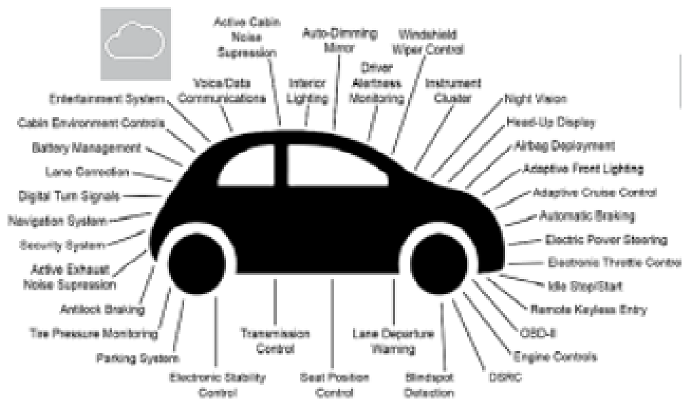
Market Trends

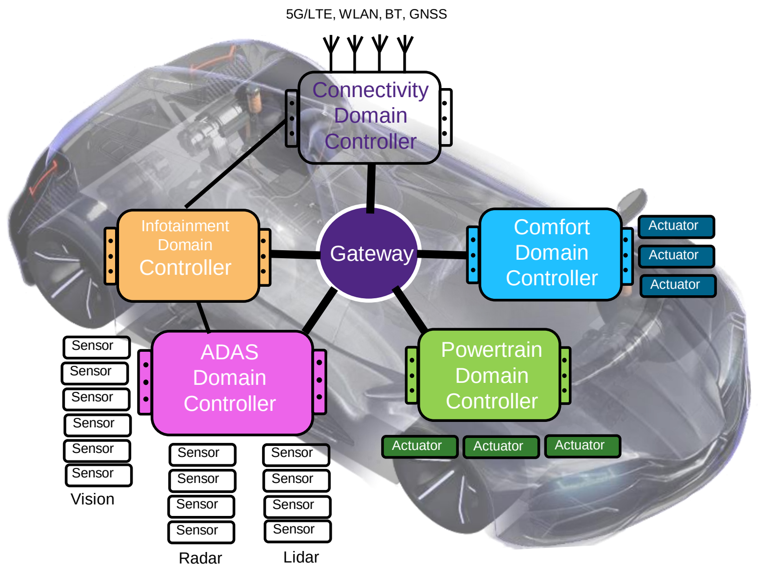
Towards Programmable Heterogeneous Multicore Architectures

A Data Centric World, Enabled by Architectural Innovation



Market Dynamics: Reshaping Automotive SoCs





Yesterday
Distributed Architecture

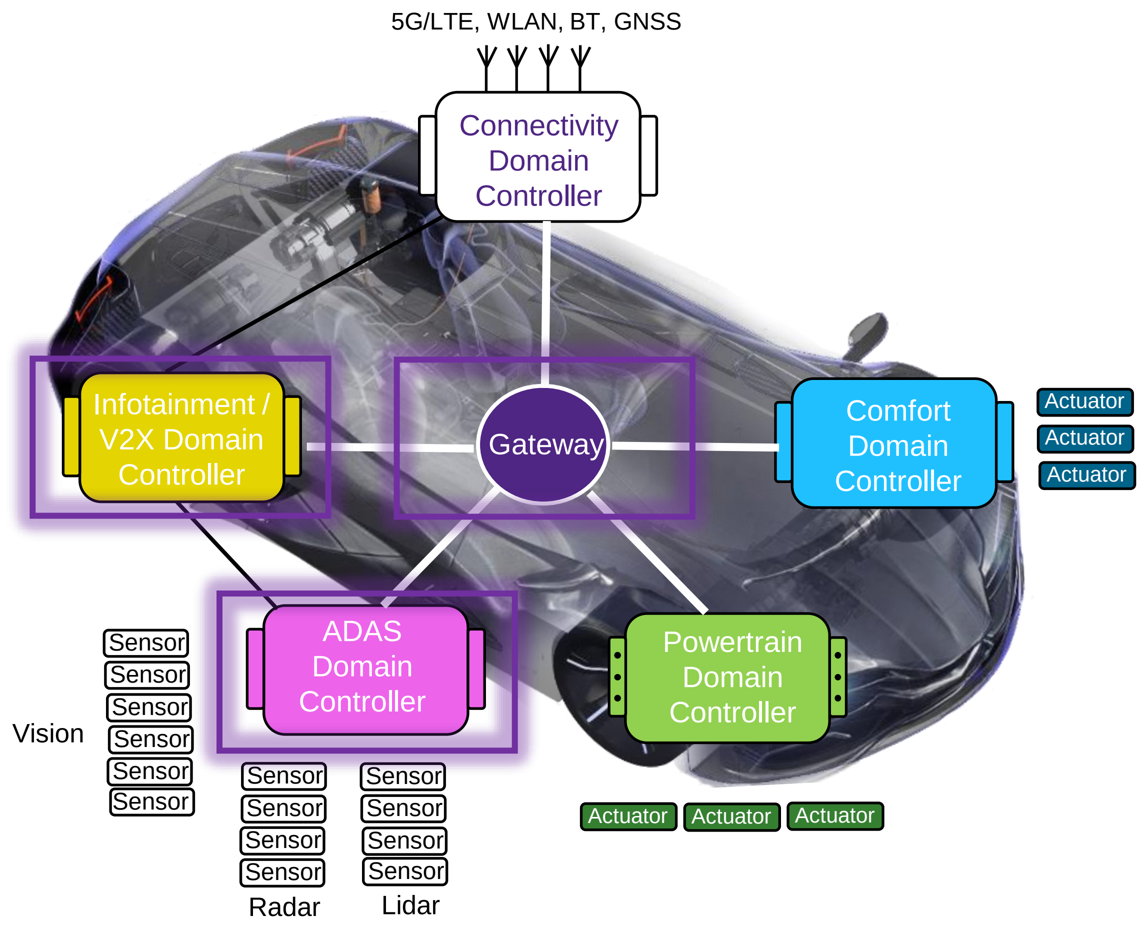
30- 100+ ECUs in a car
Mainstream MCUs

Today
Domain Logical Architecture

Consolidating of ECUs
Integration of Functions, AI & ICs

Domain Architecture Implications

It's All About Data Processing, In Time, Within Power Budget

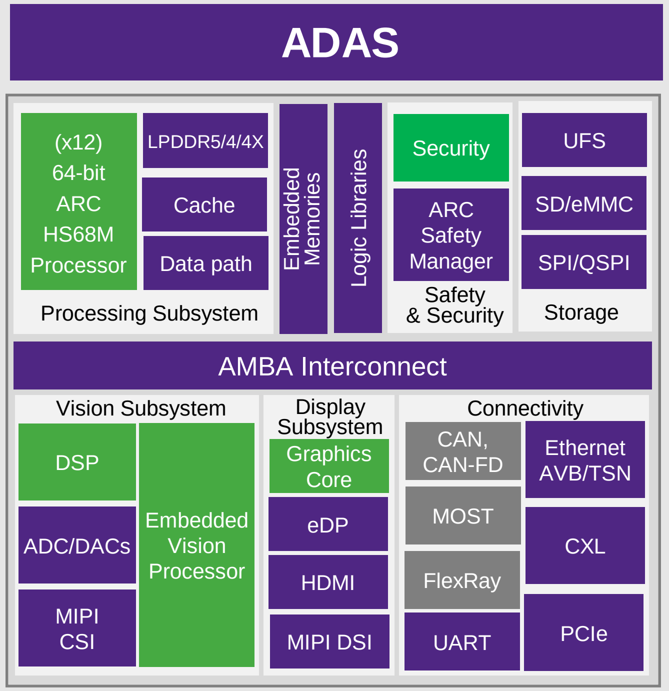


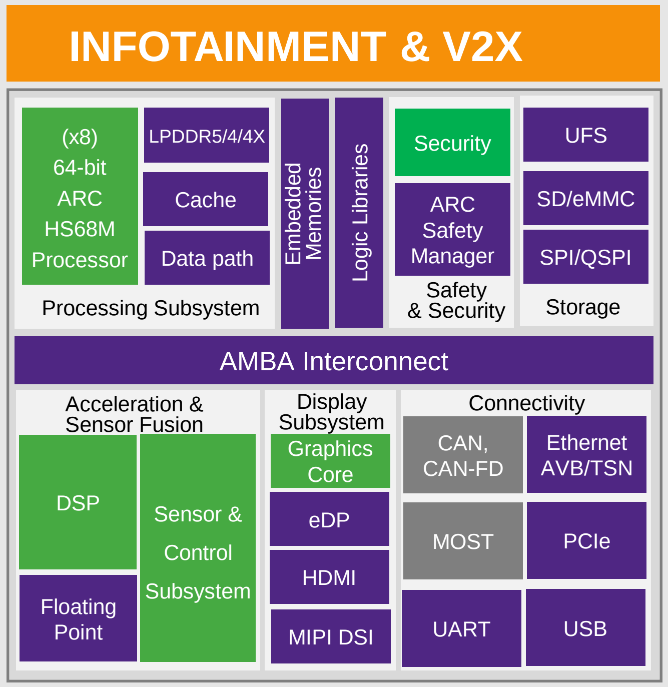
- Number/type of sensors increase data traffic
  - >15G data rate[2]
  - Waymo uses 29 cameras[1]
- Number of sensors increase compute processing
  - Computing required > 1000 TOPs[2]
- Increased need for packet processing in Gateway
- Increased need for Security

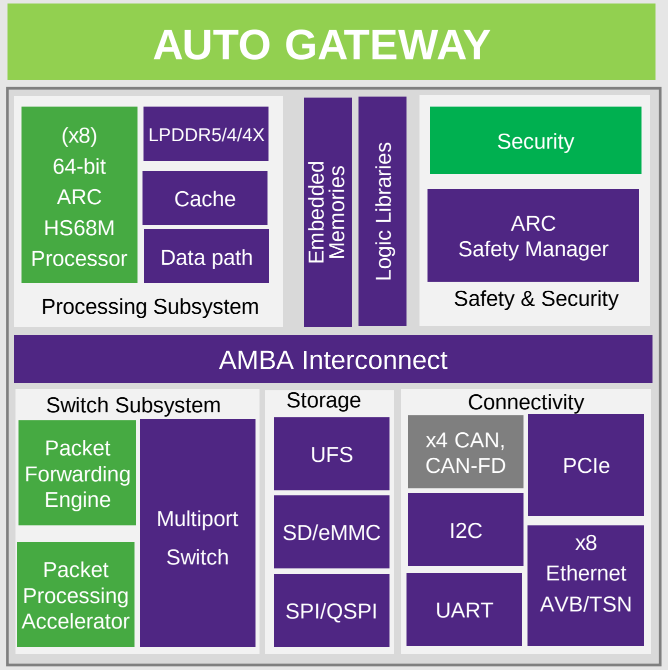
[1]Source: YooJung Ahn, Head of Design at Waymo, March, 2020
[2]Source: Yole Development, March, 2020

Automotive SoC Architectures

Heterogonous Multicore Systems, with Processors Tailored to Specific Workload





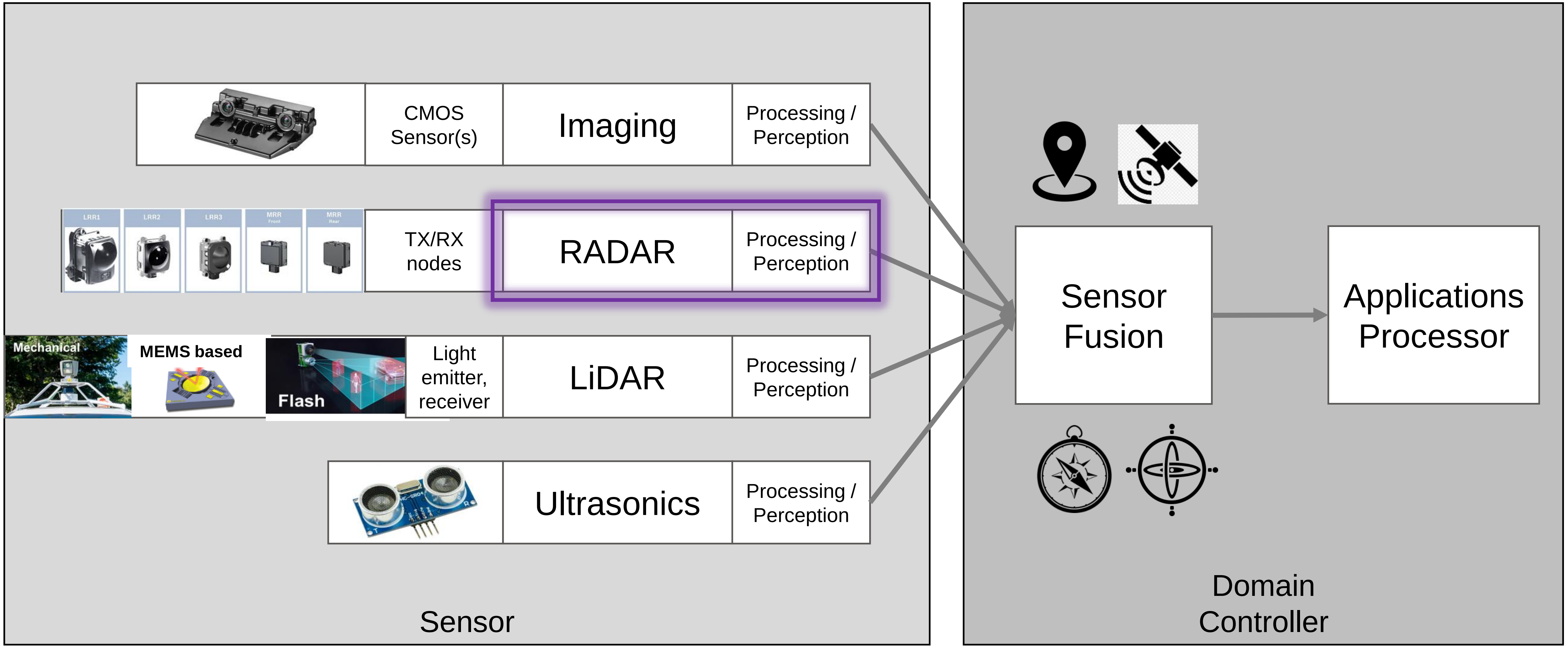


- Interfaces: LPDDR5/4/4X, Ethernet TSN, MIPI, HDMI, PCIe, CXL, ADC
- Processing: AI Accelerators, Embedded Vision, DSP, Security
- Security & SoC Safety Manager
- Sensor Fusion
- 16-/14-nm → 8-/7-nm → 5-nm
- Functional Safety

- Interfaces: LPDDR4/4X, Ethernet TSN, PCIe, DSP
- Processing: DSP, Security
- Security & SoC Safety Manager
- 5G – GPS Sensor Fusion
- 28-nm → 16-/14-nm
- Functional Safety

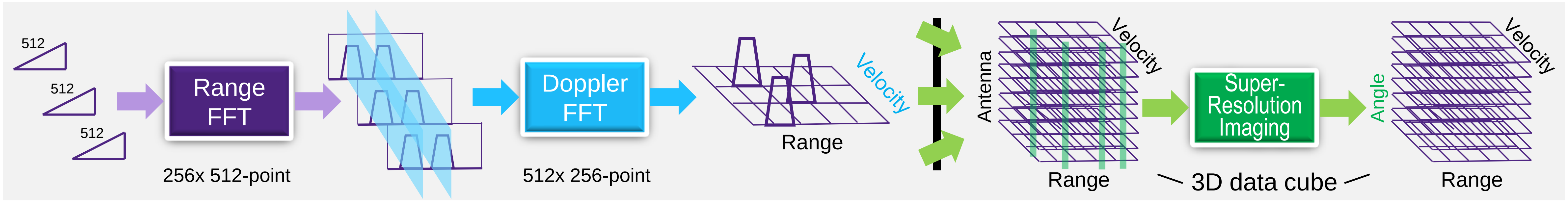
- Interfaces: LPDDR4/4X, Ethernet TSN, PCIe, DSP
- Processing: Protocol Accelerators
- Security & SoC Safety Manager
- SoC Safety Manager
- 28-nm → 16-/14-nm
- Functional Safety

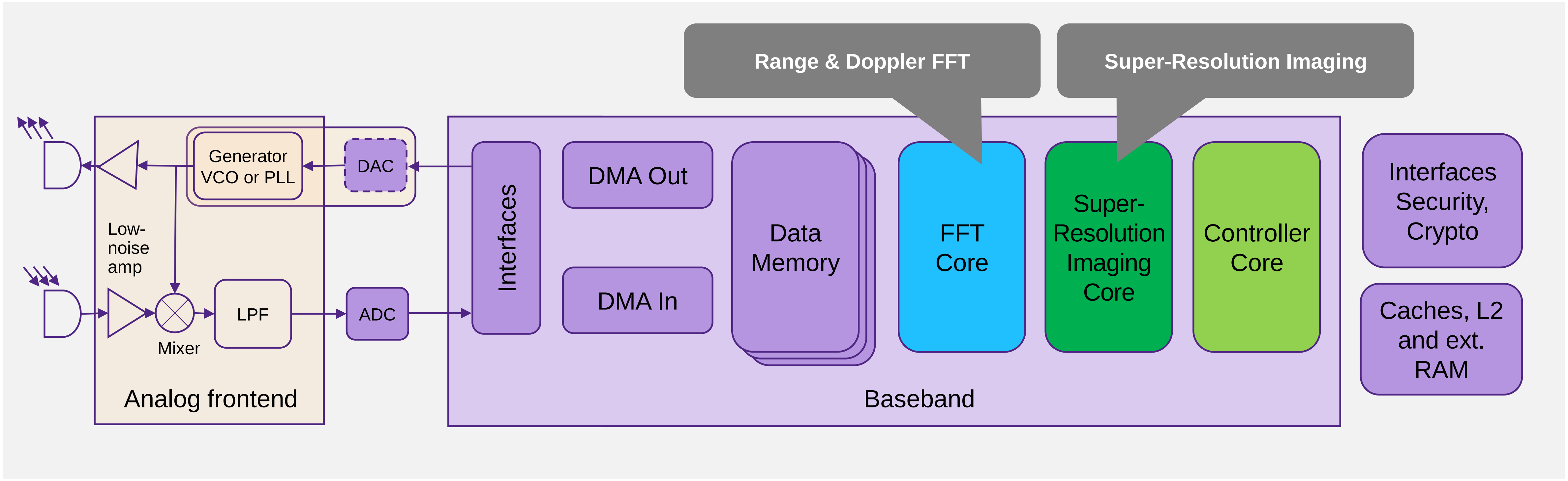
Significant Data (Pre) Processing at the Edge (Sensor)



Edge Example: RADAR

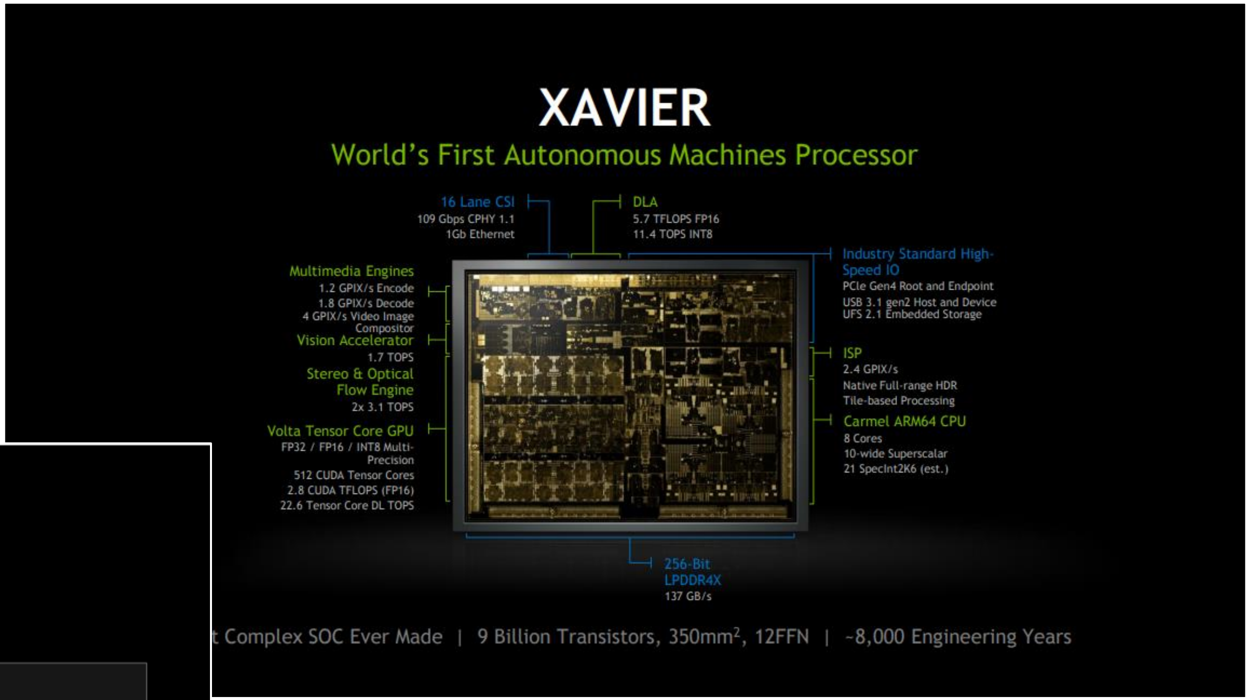
Heterogeneous Multicore Design at the Edge



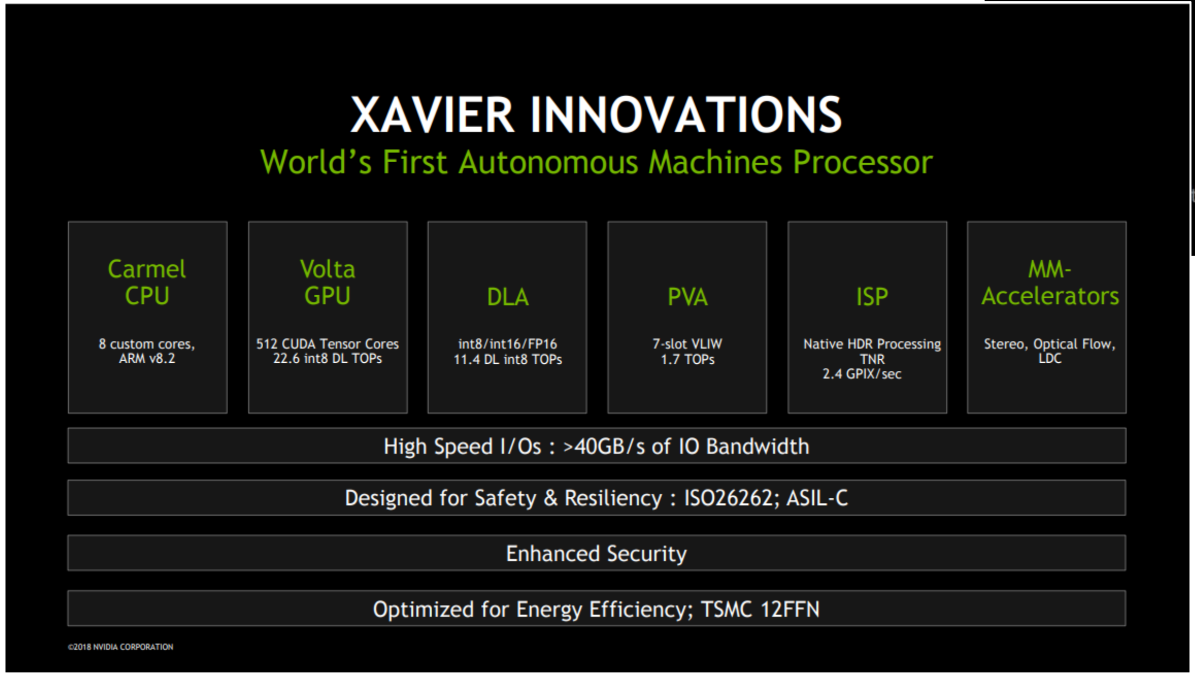


Domain Controller Example: Xavier (nvidia)

Source: nvidia (Hotchips, 2018)



Source: nvidia (Hotchips, 2018)



XAVIER INNOVATIONS

World's First Autonomous Machines Processor

Carmel CPU	Volta GPU	DLA	PVA	ISP	MM-Accelerators
8 custom cores, ARM v8.2	512 CUDA Tensor Cores 22.6 int8 DL TOPs	int8/int16/FP16 11.4 DL int8 TOPs	7-slot VLIW 1.7 TOPs	Native HDR Processing TNR 2.4 GPIX/sec	Stereo, Optical Flow, LDC

High Speed I/Os : >40GB/s of IO Bandwidth

Designed for Safety & Resiliency : ISO26262; ASIL-C

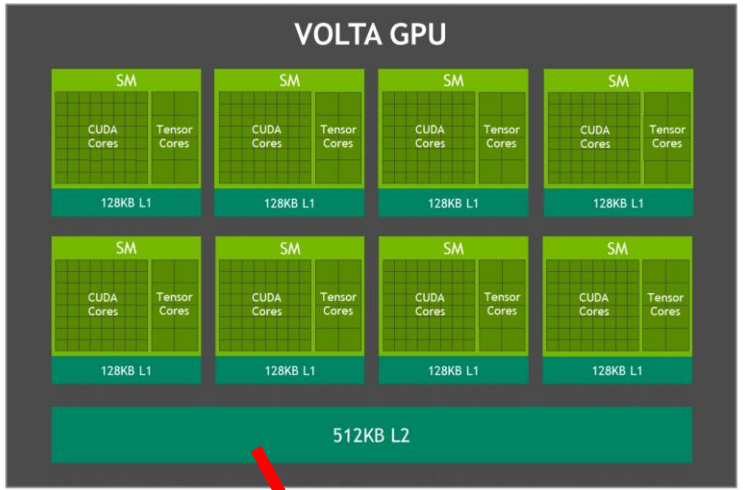
Enhanced Security

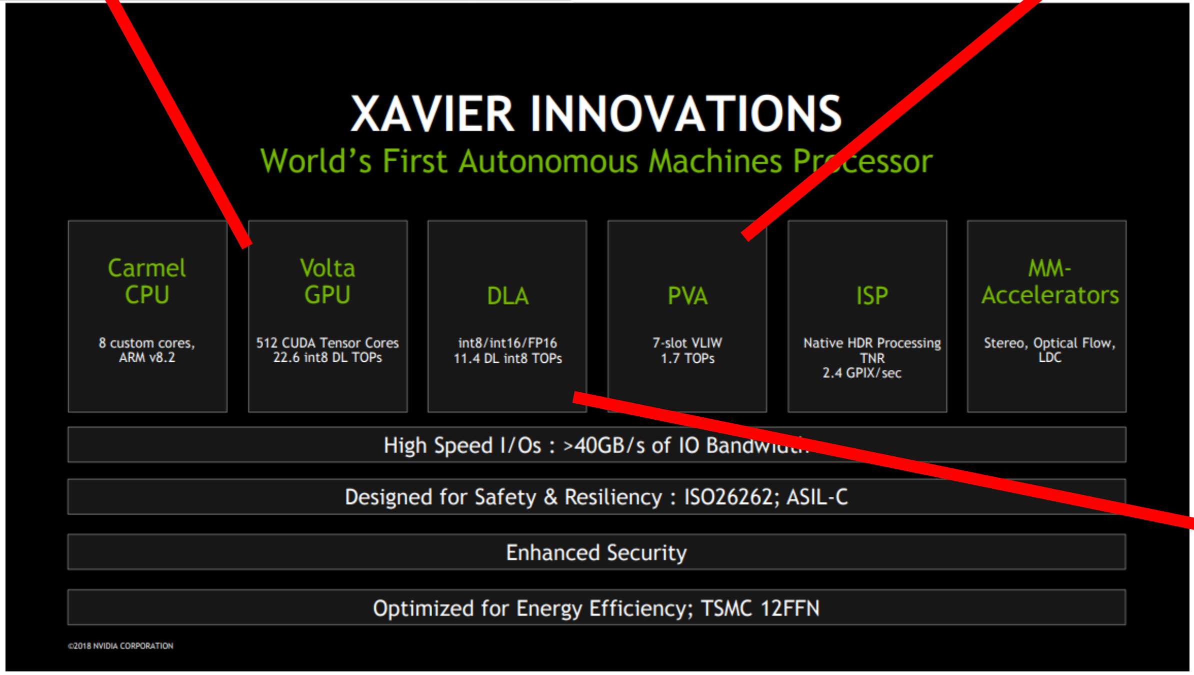
Optimized for Energy Efficiency; TSMC 12FFN

©2018 NVIDIA CORPORATION

Source: nvidia (Hotchips, 2018)

Xavier – A Heterogeneous Multicore Architecture





XAVIER INNOVATIONS

World's First Autonomous Machines Processor

Carmel CPU	Volta GPU	DLA	PVA	ISP	MM-Accelerators
8 custom cores, ARM v8.2	512 CUDA Tensor Cores 22.6 int8 DL TOPs	int8/int16/FP16 11.4 DL int8 TOPs	7-slot VLIW 1.7 TOPs	Native HDR Processing TNR 2.4 GPIX/sec	Stereo, Optical Flow, LDC

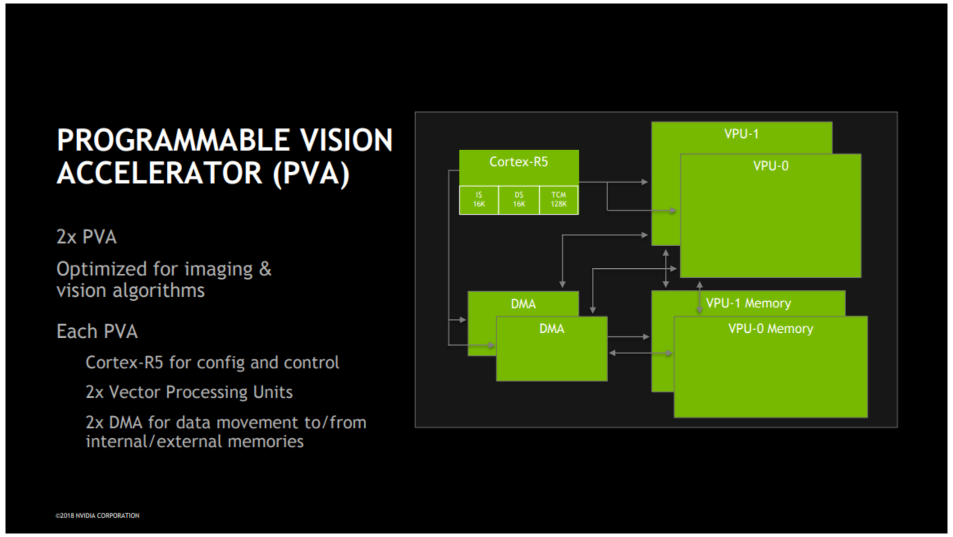
High Speed I/Os : >40GB/s of IO Bandwidth

Designed for Safety & Resiliency : ISO26262; ASIL-C

Enhanced Security

Optimized for Energy Efficiency; TSMC 12FFN

Source: nvidia (Hotchips, 2018)



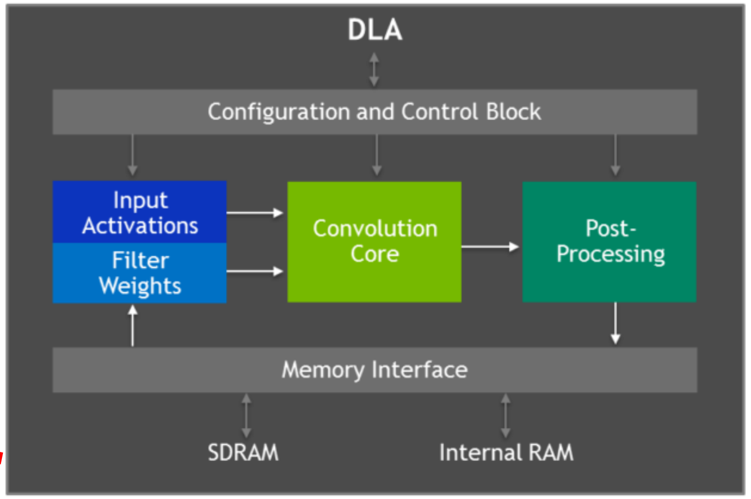
PROGRAMMABLE VISION ACCELERATOR (PVA)

2x PVA

Optimized for imaging & vision algorithms

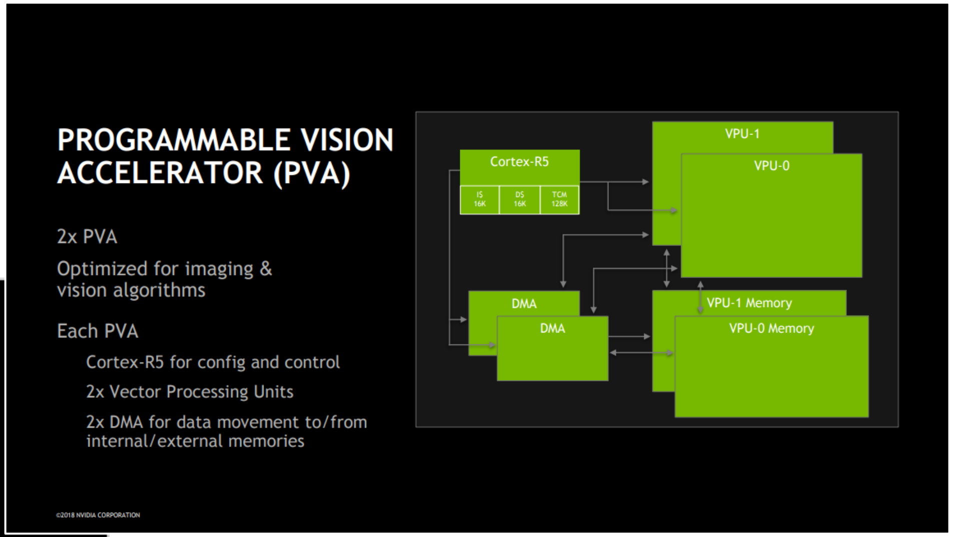
Each PVA

- Cortex-R5 for config and control
- 2x Vector Processing Units
- 2x DMA for data movement to/from internal/external memories



Source: nvidia

Inside PVA - Multicore System By Itself



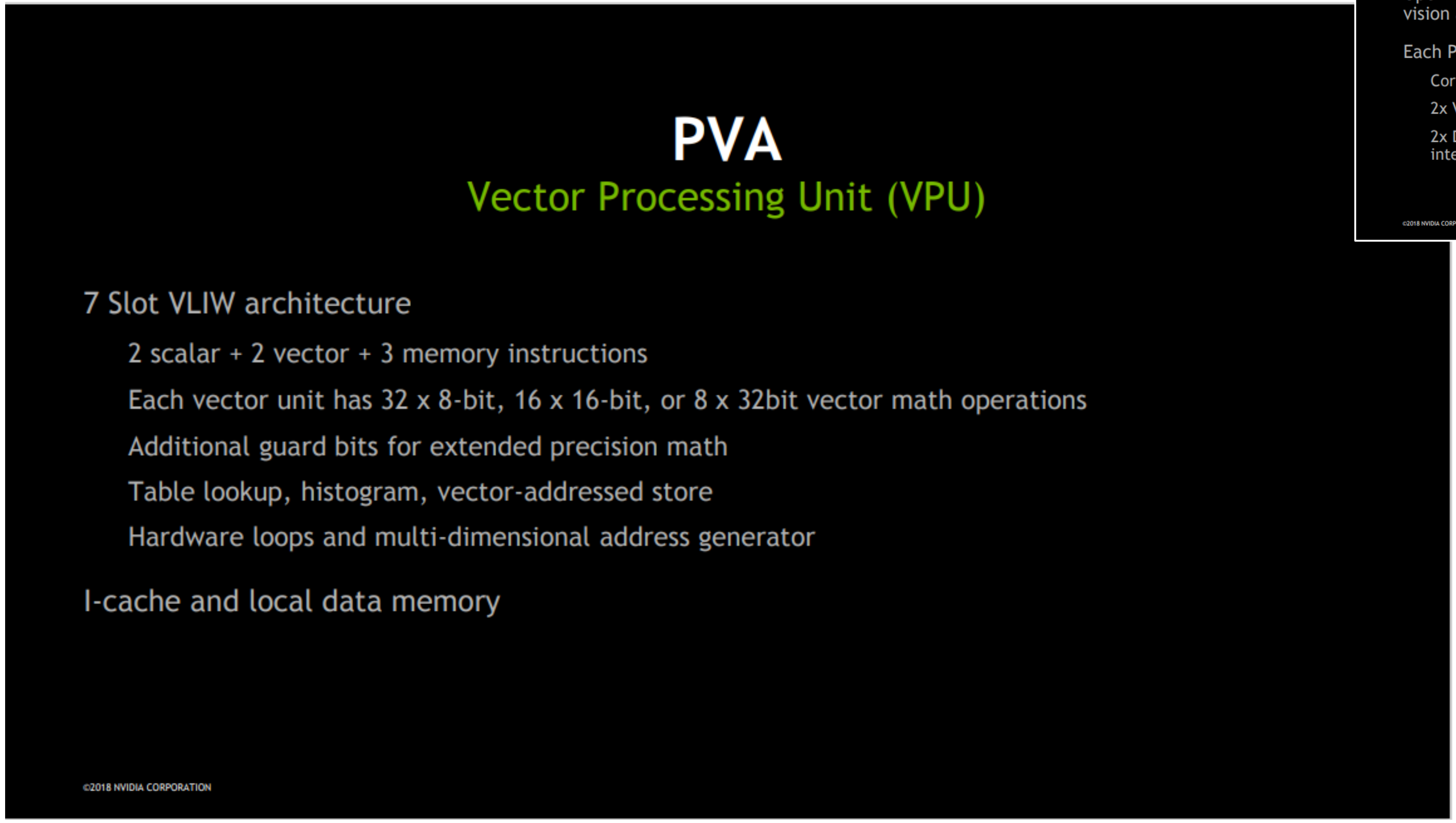
PROGRAMMABLE VISION ACCELERATOR (PVA)

2x PVA

Optimized for imaging & vision algorithms

Each PVA

- Cortex-R5 for config and control
- 2x Vector Processing Units
- 2x DMA for data movement to/from internal/external memories



PVA

Vector Processing Unit (VPU)

7 Slot VLIW architecture

- 2 scalar + 2 vector + 3 memory instructions
- Each vector unit has 32 x 8-bit, 16 x 16-bit, or 8 x 32bit vector math operations
- Additional guard bits for extended precision math
- Table lookup, histogram, vector-addressed store
- Hardware loops and multi-dimensional address generator

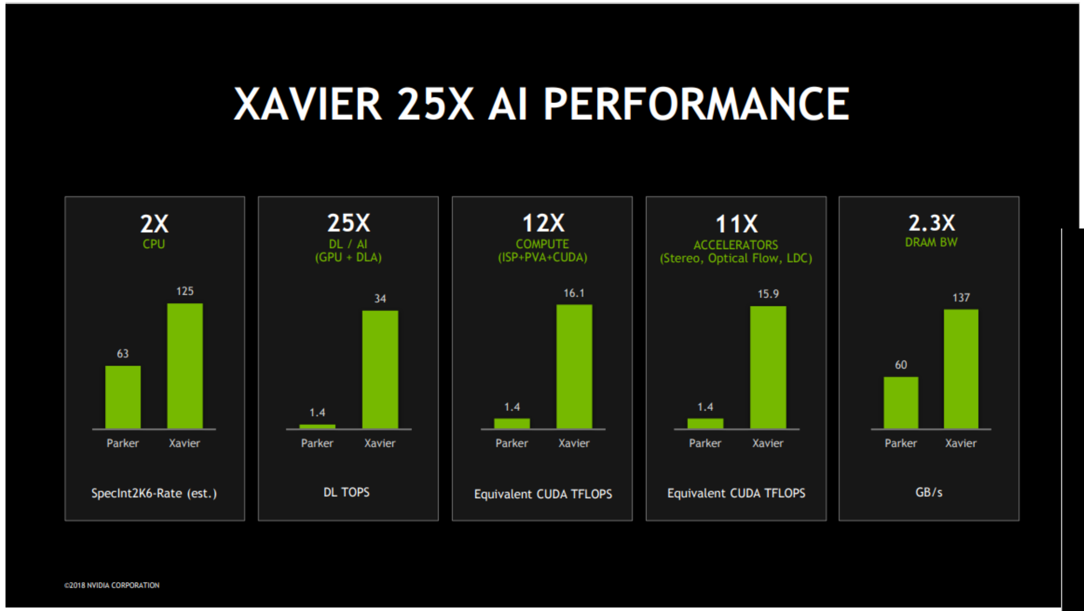
I-cache and local data memory

Tailored to the application-specific needs
- Data types
- Operations
- Memory access

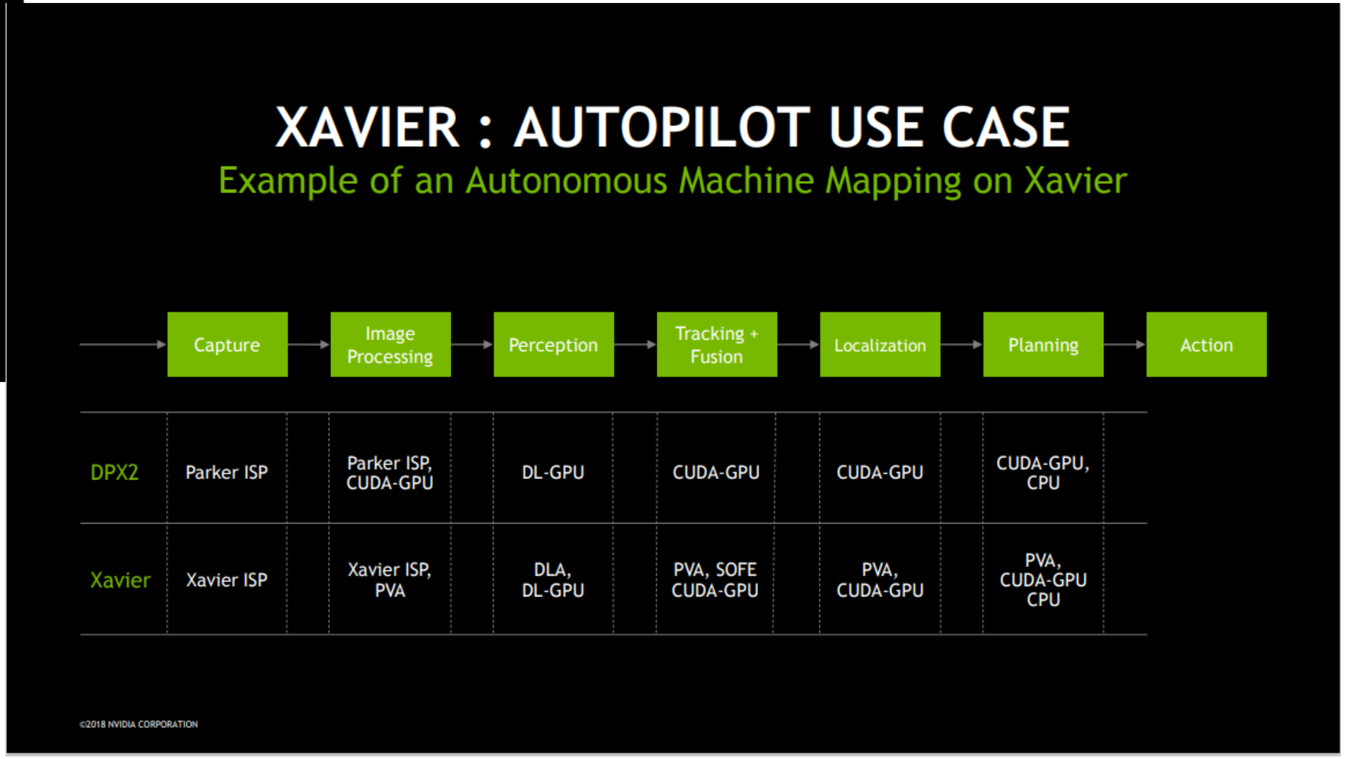
Source: nvidia (Hotchips, 2018)

What to Gain from Processor Cores Tailored to Specific Workloads

Processor Cores Tailored to Specific Workloads



Source: nvidia (Hotchips, 2018)



XAVIER : AUTOPILOT USE CASE

Example of an Autonomous Machine Mapping on Xavier

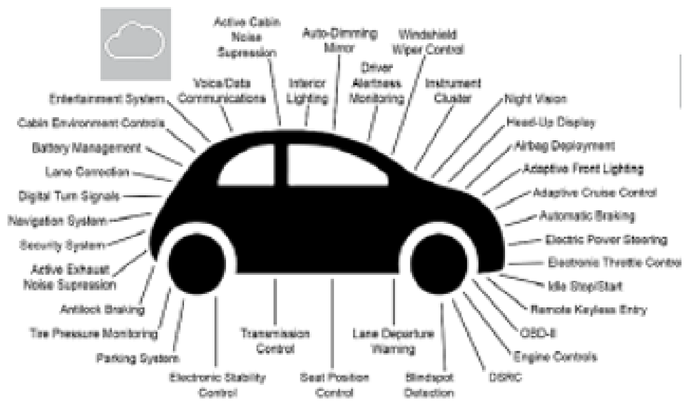
	Capture	Image Processing	Perception	Tracking + Fusion	Localization	Planning	Action
DPX2	Parker ISP	Parker ISP, CUDA-GPU	DL-GPU	CUDA-GPU	CUDA-GPU	CUDA-GPU, CPU	
Xavier	Xavier ISP	Xavier ISP, PVA	DLA, DL-GPU	PVA, SOFE CUDA-GPU	PVA, CUDA-GPU	PVA, CUDA-GPU CPU	

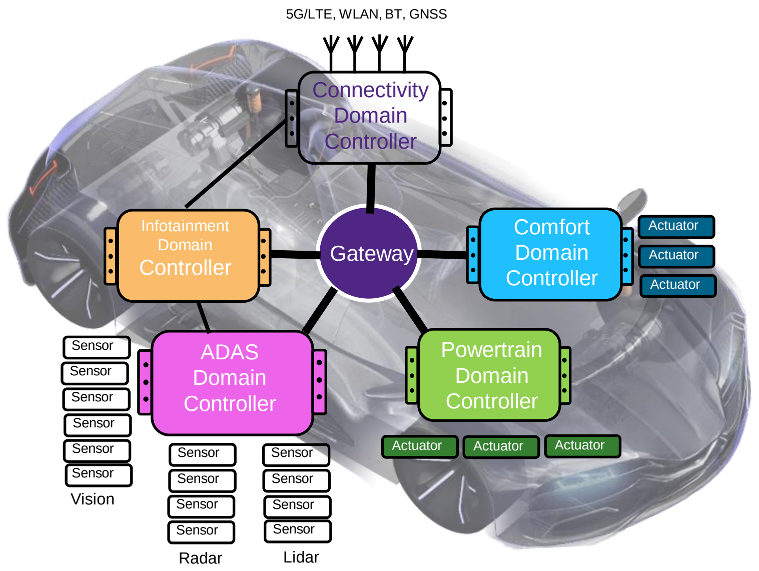
©2018 NVIDIA CORPORATION

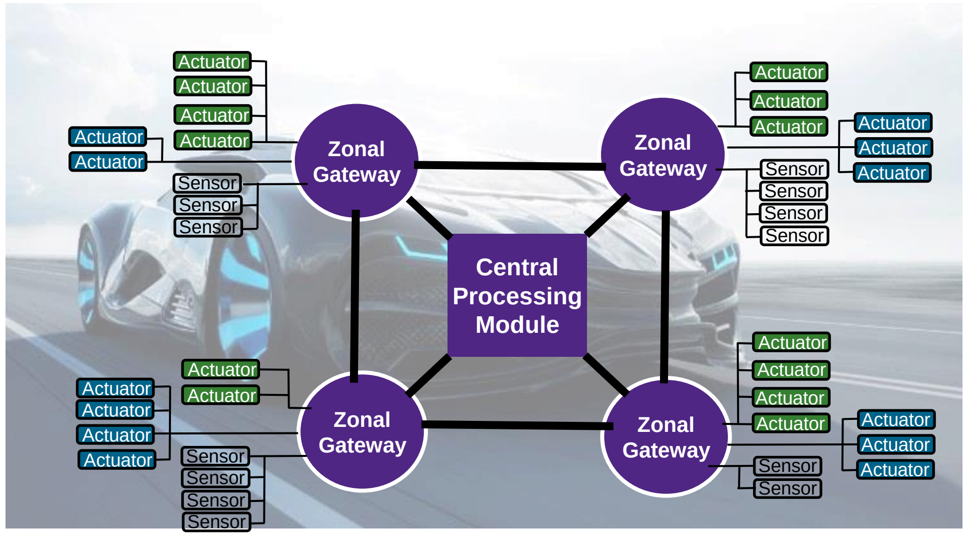
Source: nvidia (Hotchips, 2018)

Gain achieved by balancing the workload to the best-suited engine

Market Dynamics: Zonal Architecture Reshaping Automotive SoCs







Yesterday
Distributed Architecture

30- 100+ ECUs in a car
Mainstream MCUs

Today
Domain Logical Architecture

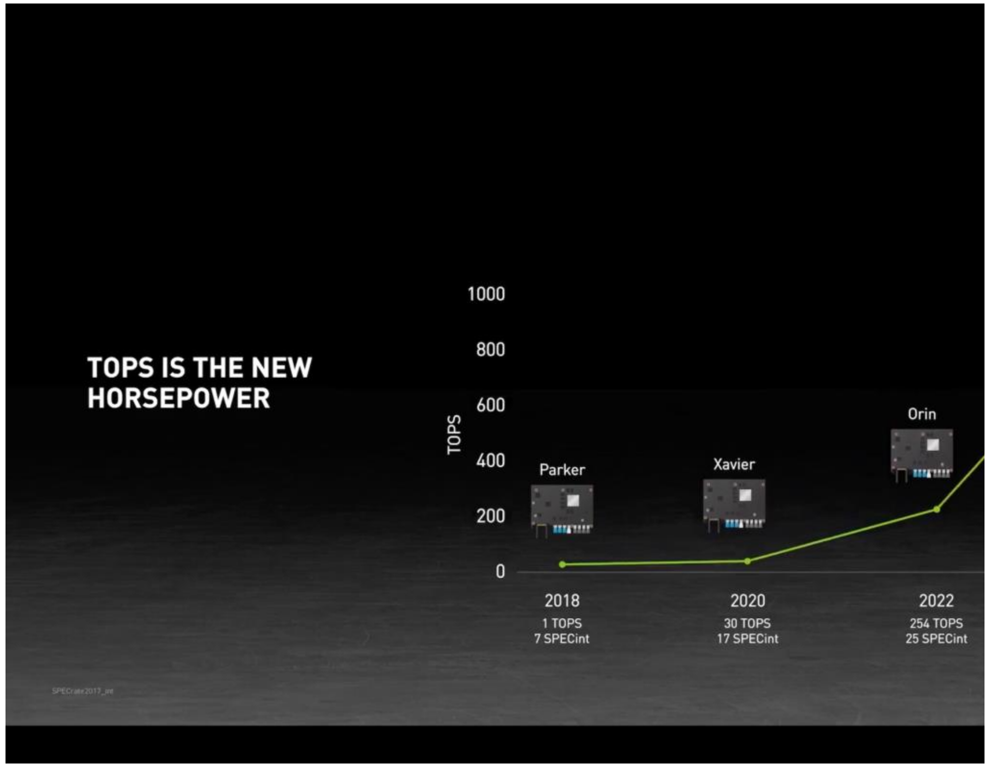
Consolidating of ECUs
Integration of Functions, AI & ICs

Tomorrow/Future
Zonal Physical Architecture

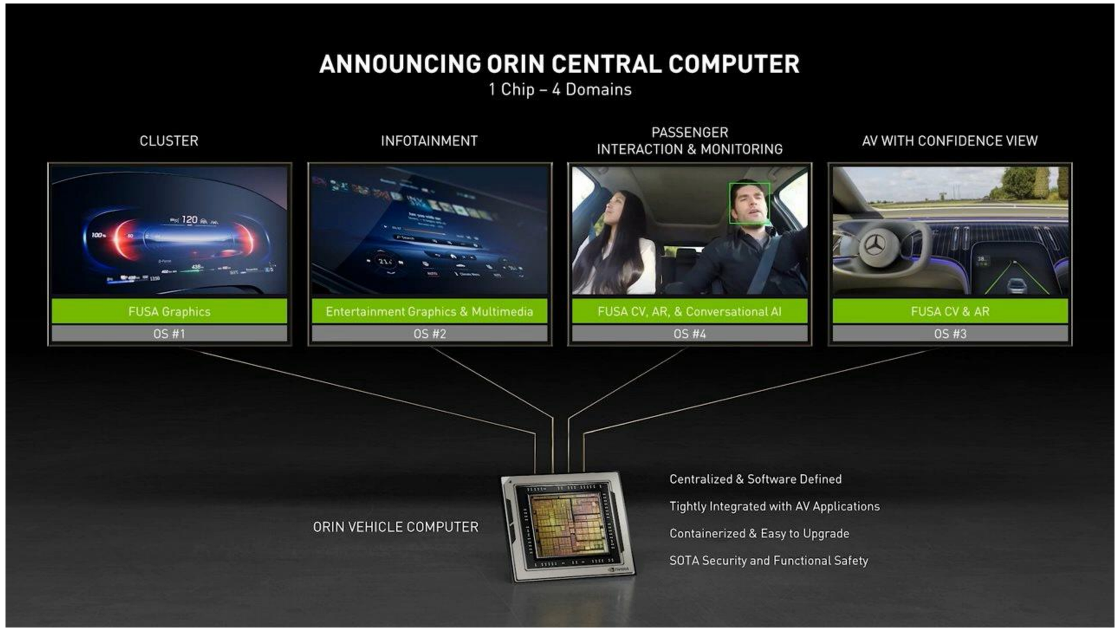
Multi-Applications Central Processing
Multi-Chip &
Higher Complexity/Performance

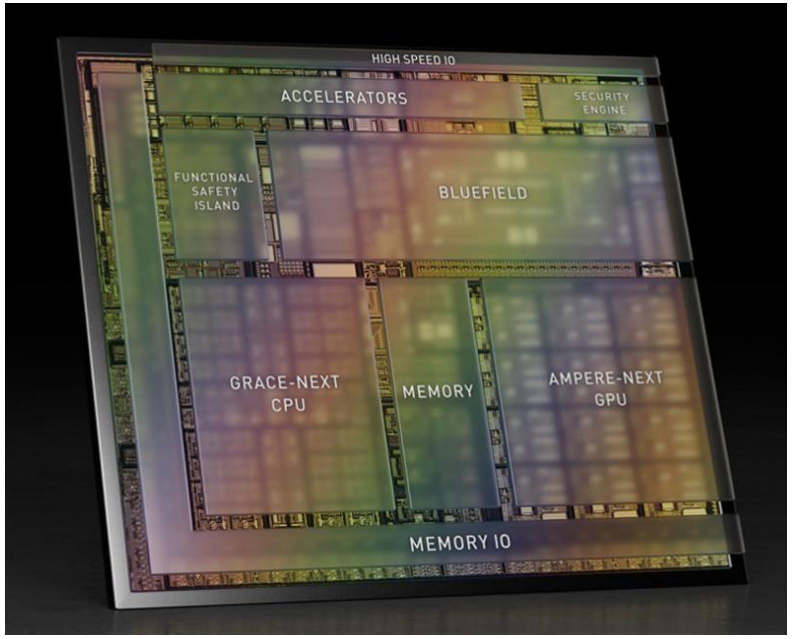
The Race Continues

Processing More Data Within the Power Budget



https://www.eenewsautomotive.com/news/nvidia-announces-new-superprocessor-autonomous-cars


ANNOUNCING ORIN CENTRAL COMPUTER
1 Chip – 4 Domains
CLUSTER
INFOTAINMENT
PASSENGER INTERACTION & MONITORING
AV WITH CONFIDENCE VIEW
FUSA Graphics
OS #1
Entertainment Graphics & Multimedia
OS #2
FUSA CV, AR, & Conversational AI
OS #4
FUSA CV & AR
OS #3
ORIN VEHICLE COMPUTER
Centralized & Software Defined
Tightly Integrated with AV Applications
Containerized & Easy to Upgrade
SOTA Security and Functional Safety

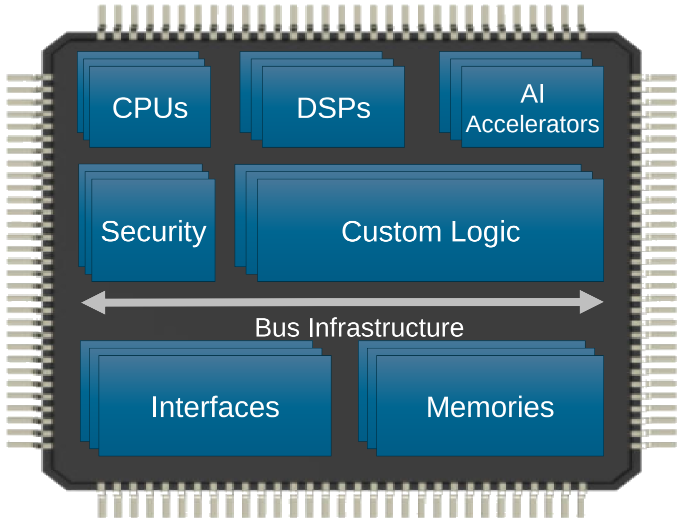


Chip Design for a Data-Centric World

- An explosion of data, triggering questions
  - Where to process the data
  - Which are the best processors and memories for different kind of data
  - How to structure, partition and prioritize the movement of raw and processed data

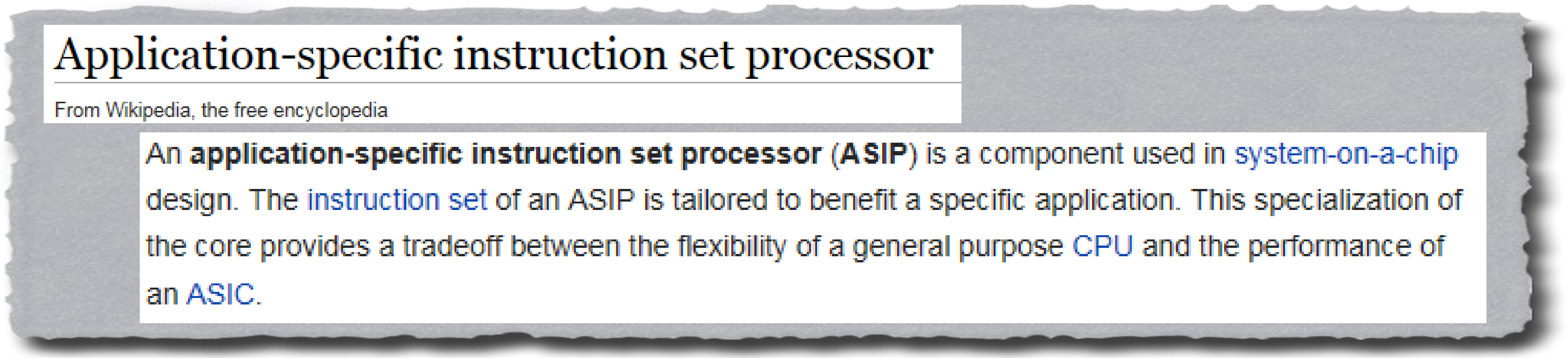
- "Golden Age" for architecture design
  - System architecture: Edge processing vs. centralized processing
  - Device architecture: 3D packaging, chiplets
  - SoC architecture: heterogenous multicore, layered memories, interconnects, …..

- SoC decisions
  - Make it programmable whenever possible, to allow for flexibility on the payload, and fixes / bypasses after tapeout
  - Select the processor(s) that fit the applications
  - Ensure efficient data exchange and synchronization between various cores
  - Define the right memory architecture, as it determines performance and power efficiency



Introduction to ASIP Designer

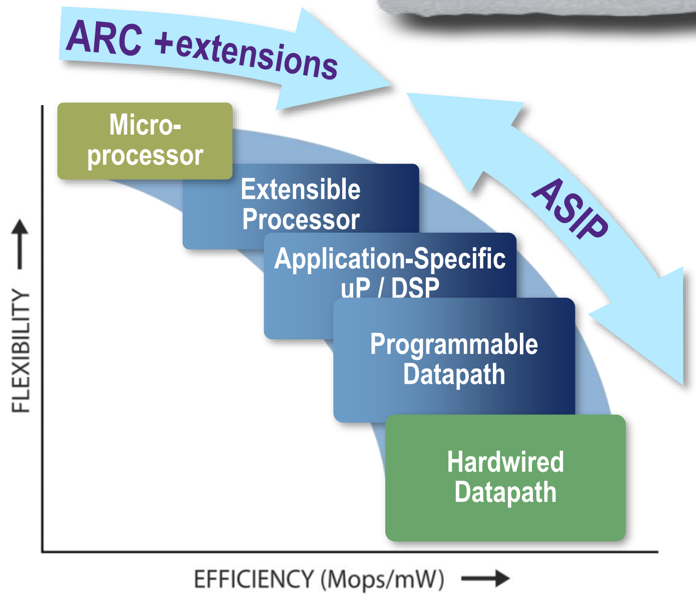
ASIP – Combining Efficiency and Flexibility



Application-specific instruction set processor

From Wikipedia, the free encyclopedia

An **application-specific instruction set processor** (**ASIP**) is a component used in system-on-a-chip design. The instruction set of an ASIP is tailored to benefit a specific application. This specialization of the core provides a tradeoff between the flexibility of a general purpose CPU and the performance of an ASIC.



Application-Specific Instruction-set Processors

ASIP Benefits: the 3 P’s

Maximize performance

- Architectural specialization
- Parallelism: instruction-level, data-level, task-level

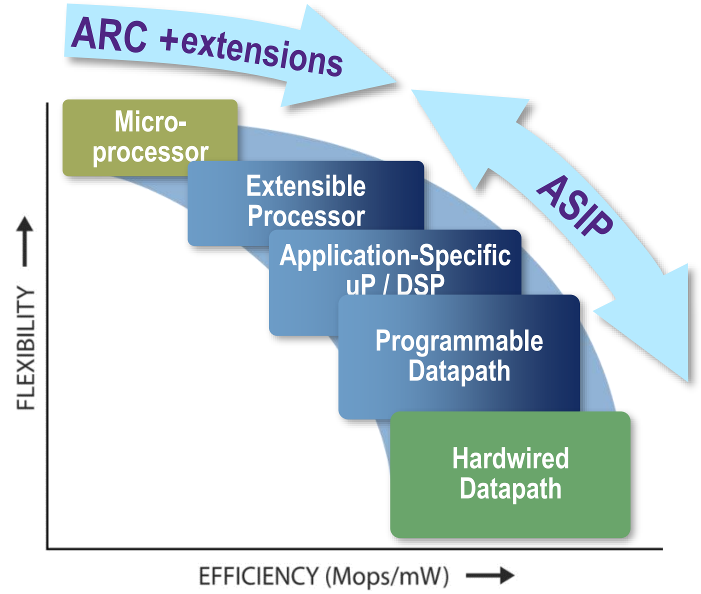
Minimize power consumption

- Architectural specialization
- Parallelism: instruction-level, data-level, task-level
- Power-optimised RTL generation
- Power-gating of cores

Programma-bility

- Support changing requirements without SoC re-spin
- Quick algorithm mapping from C to silicon, with easy debugging

- ASIPs bridge the gap between microprocessor cores and hardware

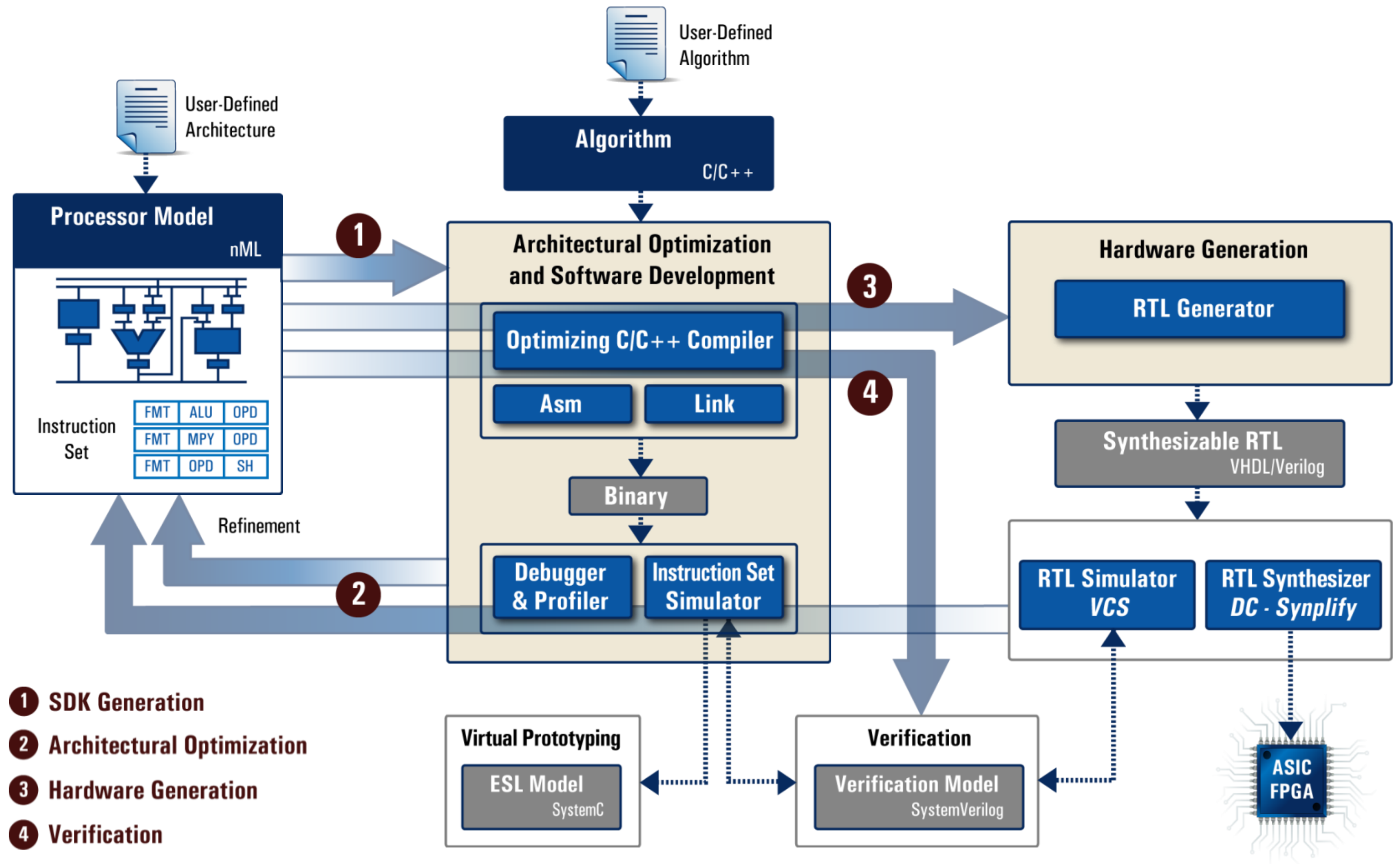


“ASIP Designer”: not licensable IP, but an EDA tool

→ Enables specialization and innovation per customer, per product

ASIP Designer

Tool Flow

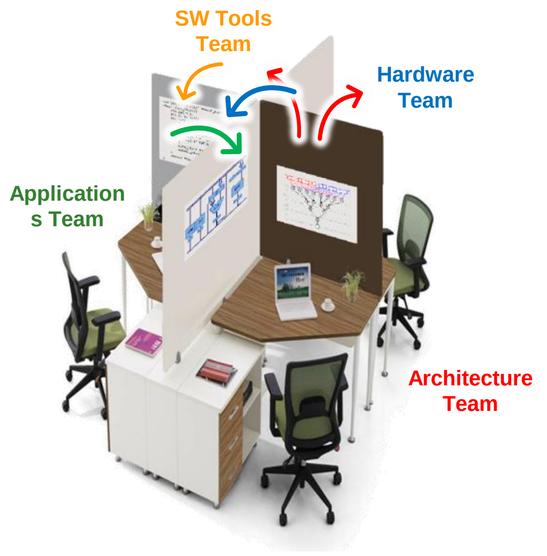


Supported design steps

- Modeling of instruction-set architectures: nML language
- Automatic generation of software development kit, including an efficient C compiler
- Algorithm-driven architectural exploration: **“Compiler-in-the-Loop”**
- Automatic generation of synthesizable RTL **“Synthesis-in-the-Loop”**
- Design verification

Processor Design is Multi-Disciplinary

Top-Down Approach



Applications Team
Rough algorithmic spec

↓

Architecture Team
Architecture spec

↙ ↘

Hardware Team
RTL coding & synthesis

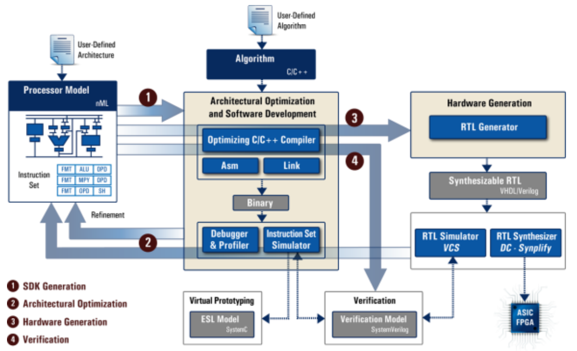
SW Tools Team
Assembler, simulator (compiler?)

↘ ↙

Applications Team
Real application code

- Largely a top-down process, with too-little / too-late feedback
- Limited multi-disciplinary skills in teams
- Inconsistent RTL & SW tools, due to misinterpretation of architecture spec
- Suboptimal performance for the eventual applications
- Too long design cycle, missed business opportunities

Agile Approach



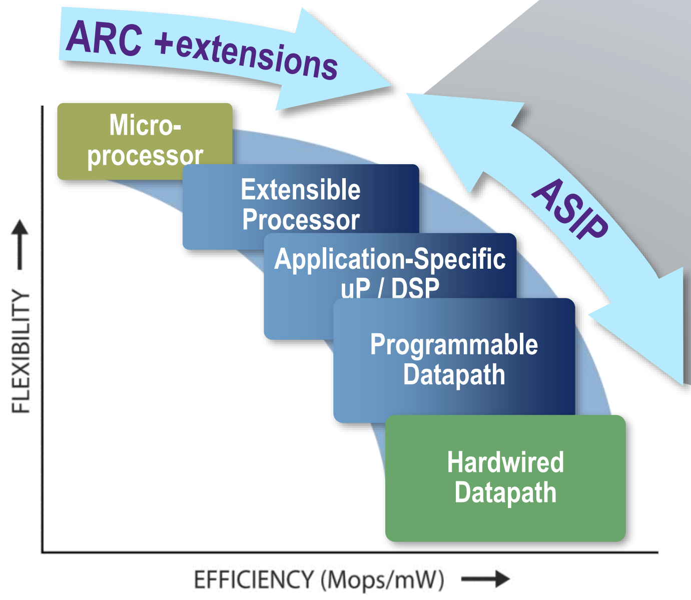
- Tool automation, spanning all design disciplines
- All design views automatically derived from single "golden model" that is continuously refined
- Tool helps designers to develop inter-disciplinary skills quickly
- Early SW development speeds up project and improves architecture
- Small design team can master sophisticated designs
- Consistency guaranteed by methodology, eases verification
- Short design cycles, easy creation of design variants

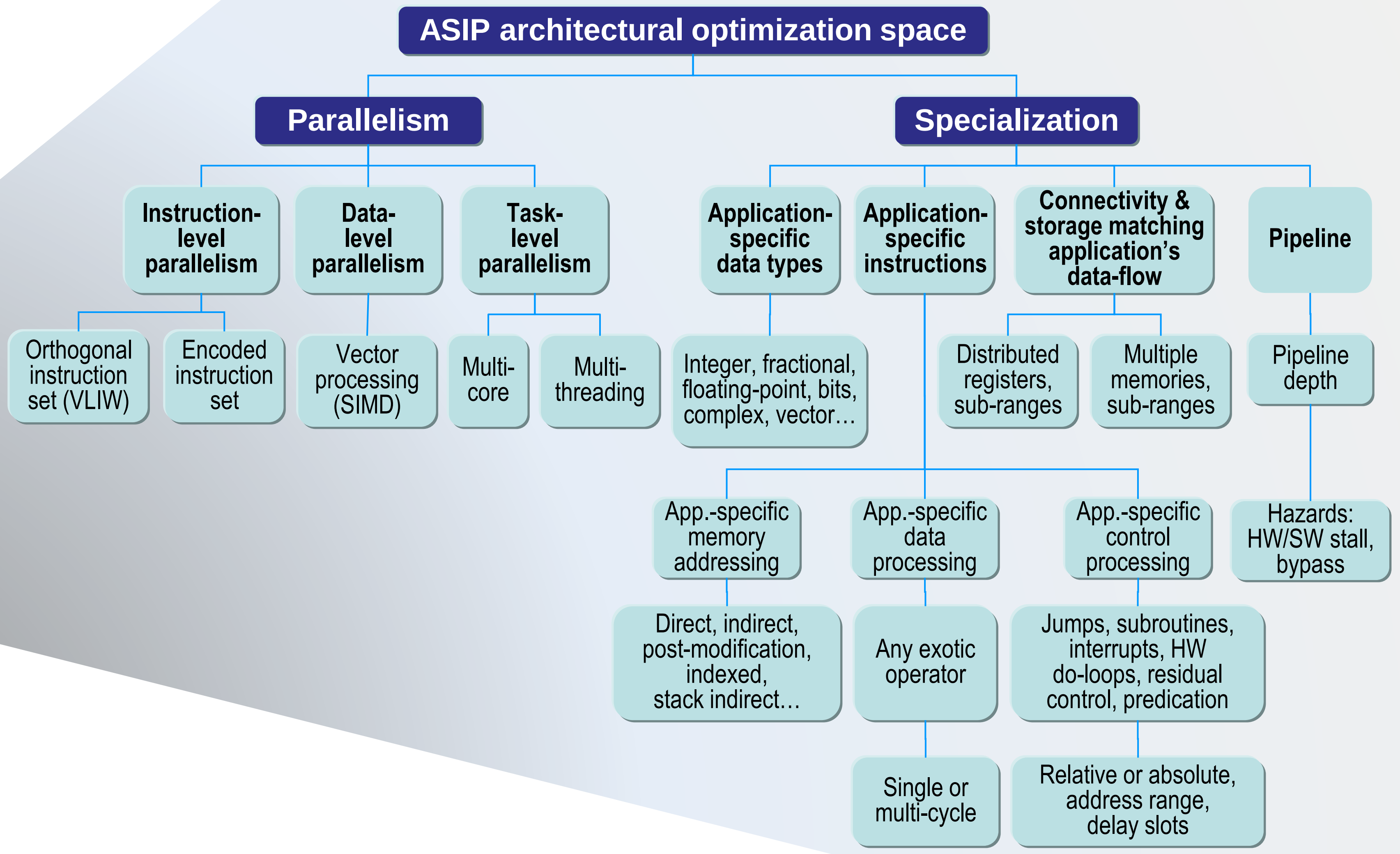
ASIP Designer

Broad Architectural Scope

nML and ASIP Designer…

- Support a wide range of ASIP architectures, beyond configurable templates
- Enable true architectural exploration



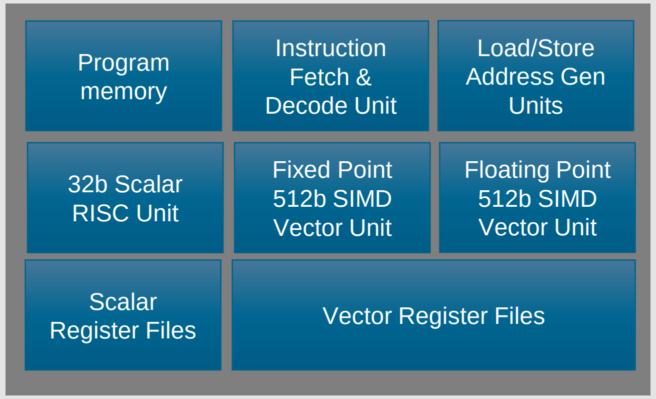

ASIP architectural optimization space
Parallelism
Specialization
Instruction-level parallelism
Data-level parallelism
Task-level parallelism
Application-specific data types
Application-specific instructions
Connectivity & storage matching application's data-flow
Pipeline
Orthogonal instruction set (VLIW)
Encoded instruction set
Vector processing (SIMD)
Multi-core
Multi-threading
Integer, fractional, floating-point, bits, complex, vector…
Distributed registers, sub-ranges
Multiple memories, sub-ranges
Pipeline depth
App.-specific memory addressing
App.-specific data processing
App.-specific control processing
Hazards: HW/SW stall, bypass
Direct, indirect, post-modification, indexed, stack indirect…
Any exotic operator
Jumps, subroutines, interrupts, HW do-loops, residual control, predication
Single or multi-cycle
Relative or absolute, address range, delay slots

ASIP Designer – Some Case Studies

Full Architectural Freedom, Not Based on a Template

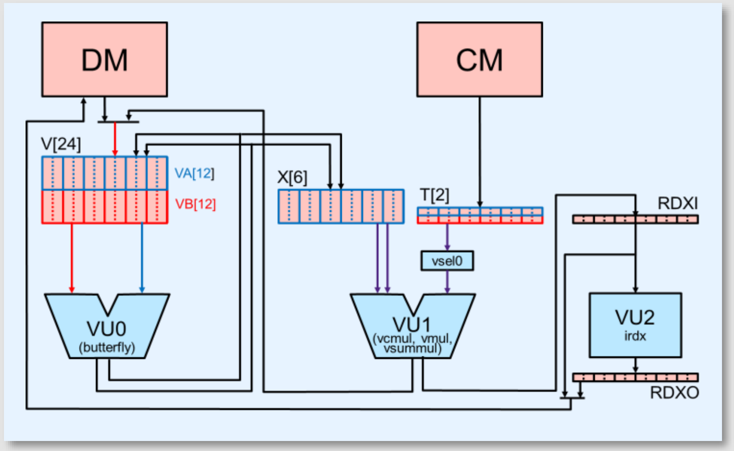
Vector-DSP

- Combined SIMD / VLIW design
- Broad data type support: Int 4/8/16/32 Float, Bfloat 16, fixed-point, complex
- Fully C-programmable


Program memory
Instruction Fetch & Decode Unit
Load/Store Address Gen Units
32b Scalar RISC Unit
Fixed Point 512b SIMD Vector Unit
Floating Point 512b SIMD Vector Unit
Scalar Register Files
Vector Register Files

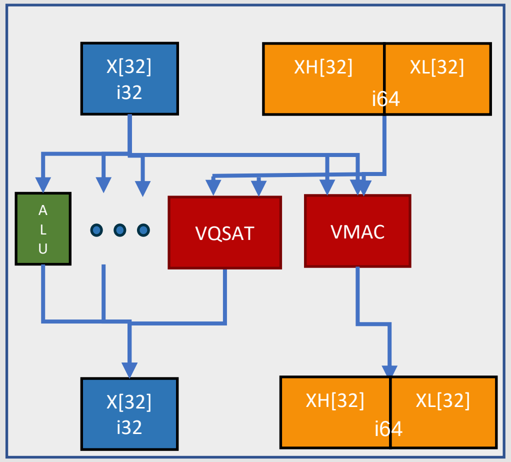
FFT / DFT Accelerator

- FFT: all power-of-2 sizes from 8 to 2048, algorithm up to radix-8
- DFT: all prime-factorizable sizes from 6 to 1536
- 8-lane SIMD, 5-way VLIW



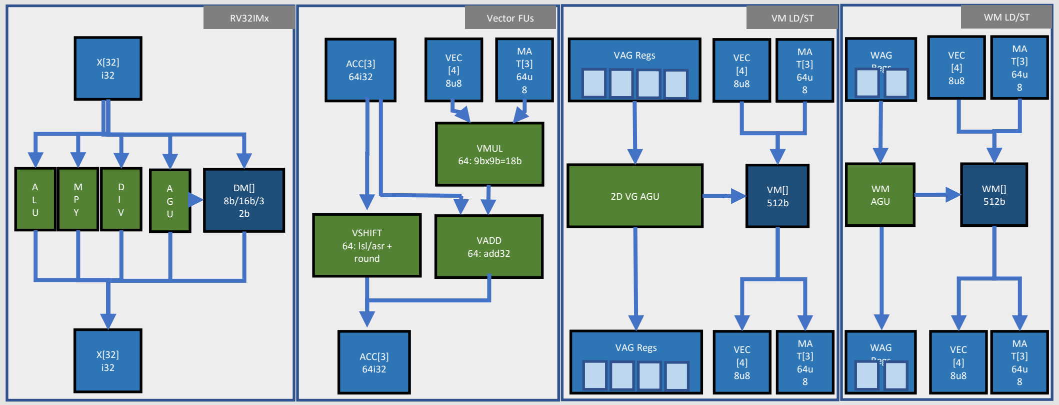
RISC-V Derivative

- Standard RISC-V ISA
- Extended with specialized instructions for keyword spotting
- 10x acceleration, 1.16x area increase
- Packed SIMD, 8-bit data type support



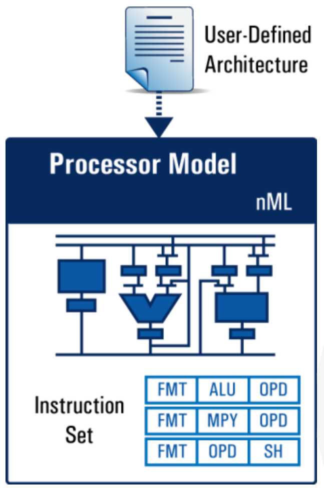
AI Accelerator

RISC-V scalar unit, dedicated vector processing and address generation units



ASIP Designer

Example Models



Name	Description
Microcontrollers	
Tnano	16-bit microcontroller, lightweight and configurable
Tmicro	16-bit microcontroller, fully featured
DLX (family)	Variants of 32-bit microcontroller
Tmcu	32-bit microcontroller
Trv32 (family)	32-bit microcontroller featuring RISC-V ISA, 3 or 5 pipeline stages (incl SDX option for **S**imple **D**atapath e**X**tensions)
Trv64 (family)	64-bit microcontroller featuring RISC-V ISA, 3 or 5 pipeline stages
PD_Triop	32-bit microcontroller with 64-bit address spaces
DSPs and generic parallel processors	
Tdsp	16/32-bit DSP
Tvec (family)	Variants of SIMD processor
Tvliw (family)	Variants of VLIW processor
Domain-specific processors	
Tmoby	Accelerator for AI application, featuring MobileNet V3 graph
Tvox	Accelerator for SLAM (simultaneous localization and mapping)
MMSE	Minimum Mean Square Error Equalization
Tgauss	Vectorization and memory management for image processing
Tmotion	Accelerator of motion estimation kernel
Tcom8	SIMD processor optimized for some communication kernels
FFTcore	Scalar implementation of complex FFT
Mxcore	Matrix processing ASIP for communication kernels
Primecore	SIMD implementation of prime-factor algorithm for FFT & DFT
JEMA, JEMB	Dual ASIP for JPEG encoding (accelerating DCT, VLC)

Many examples provided

- Microcontrollers
- DSPs
- SIMD
- VLIW
- Multi-threading
- Domain specific

Examples are good starting points

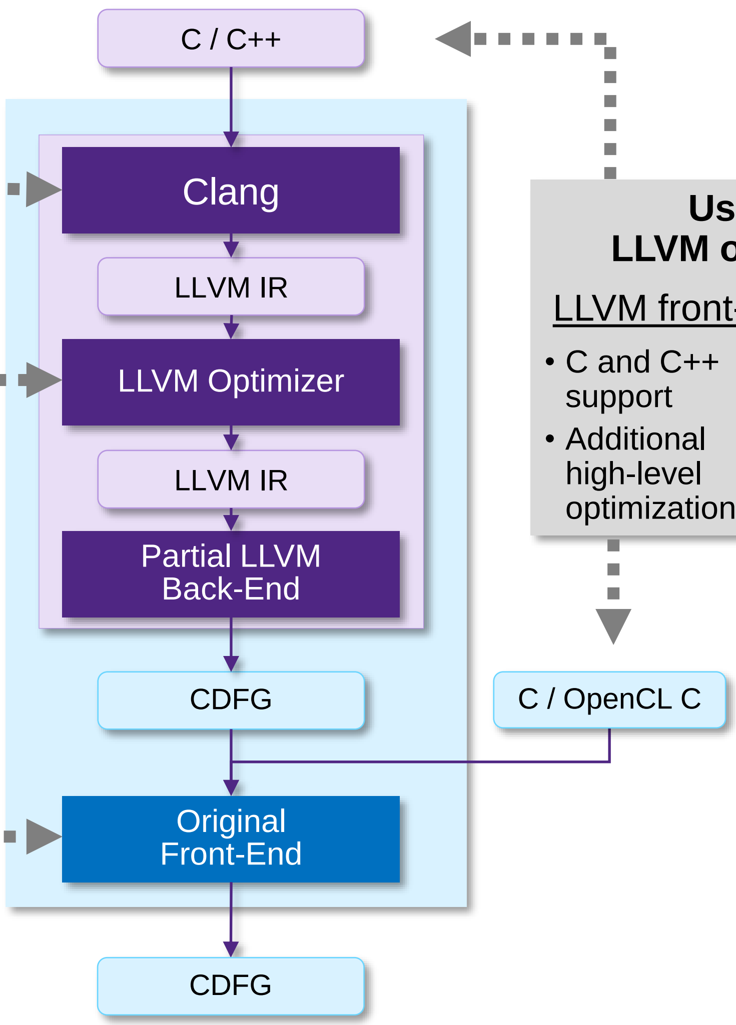
- Get a jump start with known working example
- Learn about modelling concepts
- Provided in nML source code, users can modify for application-specific tuning

Retargetable Compiler

Language Front End

Extended to support ASIP architectures
- Datatypes
- Memory model
- DSP optim.

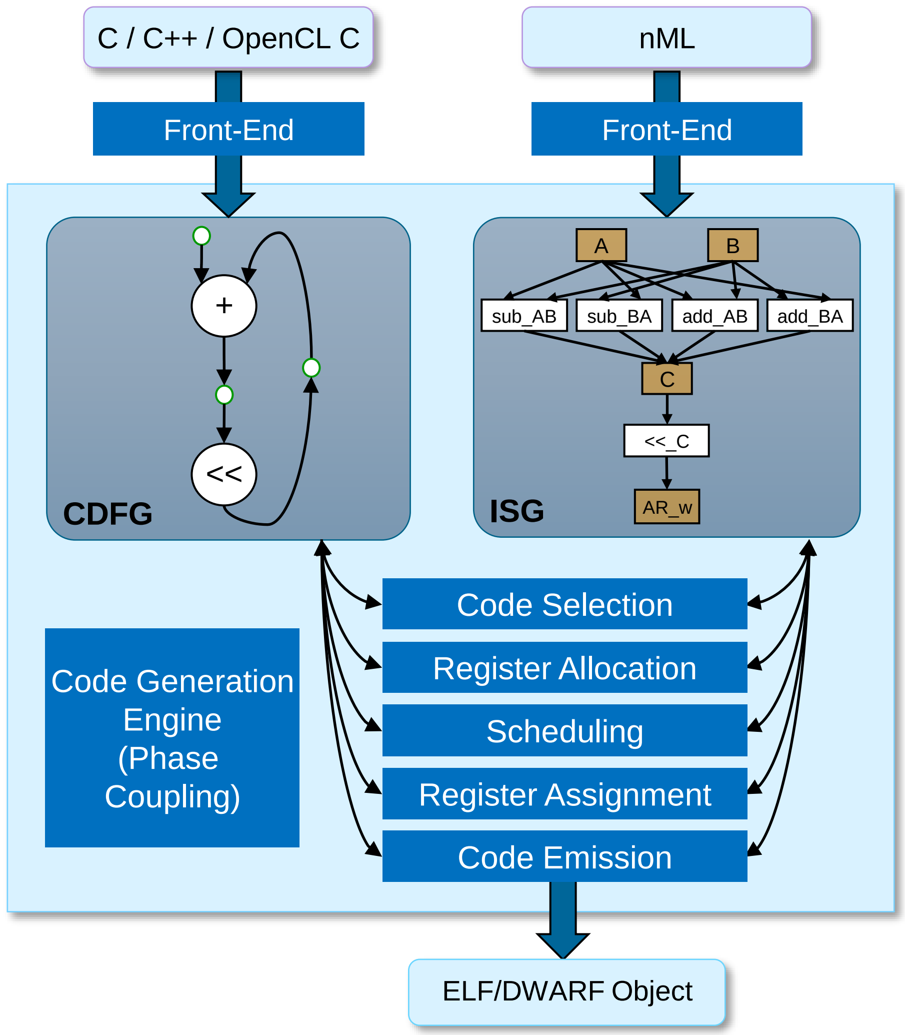
Combine high-level optimizations from LLVM and original front-end



User can choose: LLVM or original front-end

LLVM front-end	Original front-end
• C and C++ support	• C and OpenCL C support
• Additional high-level optimizations	• Better user-control: preferred during archit. exploration

Retargetable Back End

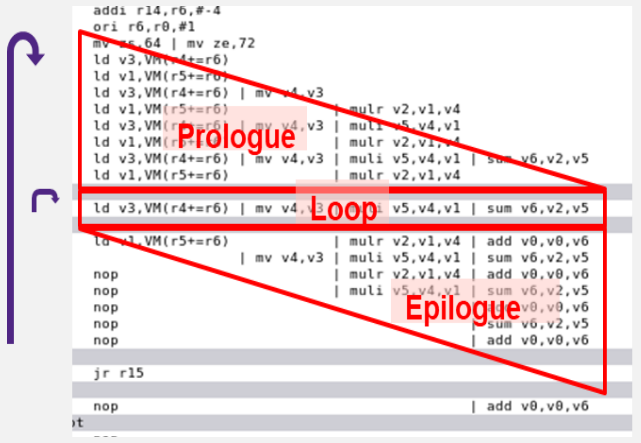


ASIP Designer SDK

Compiler Back End and Parallel DSP Architectures

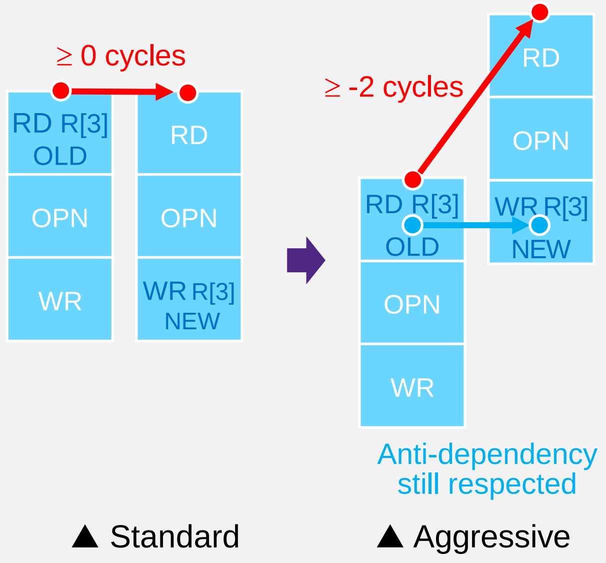
- Target architectures
  - ILP with multiple slots
    - E.g. 5 slots, typically heterogeneous
  - Wide SIMD
  - Central or distributed register file(s)
    - Limited capacity
  - Deep pipeline
    - Typically, ~10 stages, with 5 stages for the execute pipe

- Compiler optimizations
  - VLIW Scheduling including software pipelining for inner and higher-level loops

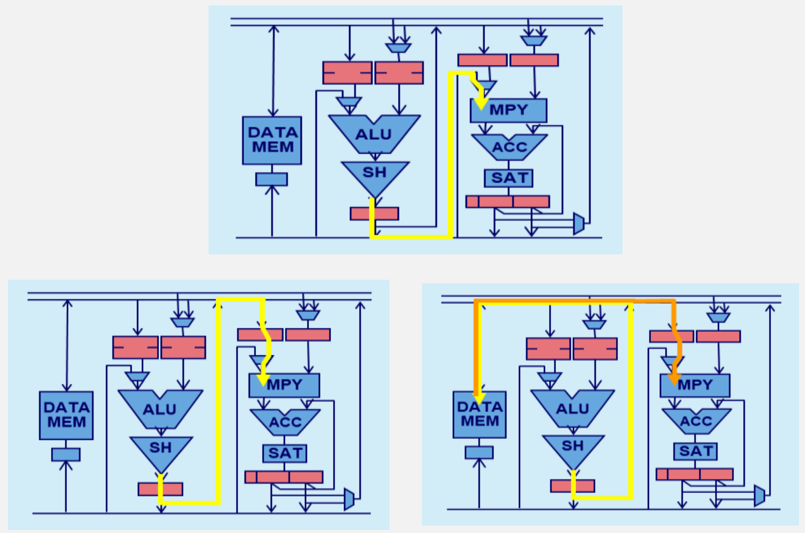


```
addi r14,r6,#-4
ori r6,r0,#1
mv zs,64 | mv ze,72
ld v3,VM(r4+=r6)
ld v1,VM(r5+=r6)
ld v3,VM(r4+=r6) | mv v4,v3
ld v1,VM(r5+=r6)             | mulr v2,v1,v4
ld v3,VM(r4+=r6) | mv v4,v3  | muli v5,v4,v1
ld v1,VM(r5+=r6)             | mulr v2,v1,v4
ld v3,VM(r4+=r6) | mv v4,v3  | muli v5,v4,v1 | sum v6,v2,v5
ld v1,VM(r5+=r6)             | mulr v2,v1,v4

ld v3,VM(r4+=r6) | mv v4,v3  | muli v5,v4,v1 | sum v6,v2,v5

ld v1,VM(r5+=r6)             | mulr v2,v1,v4 | add v0,v0,v6
                 | mv v4,v3  | muli v5,v4,v1 | sum v6,v2,v5
nop                          | mulr v2,v1,v4 | add v0,v0,v6
nop                          | muli v5,v4,v1 | sum v6,v2,v5
nop                                          | add v0,v0,v6
nop                                          | sum v6,v2,v5
nop                                          | add v0,v0,v6

jr r15

nop                                          | add v0,v0,v6
```

  - Aggressive scheduling

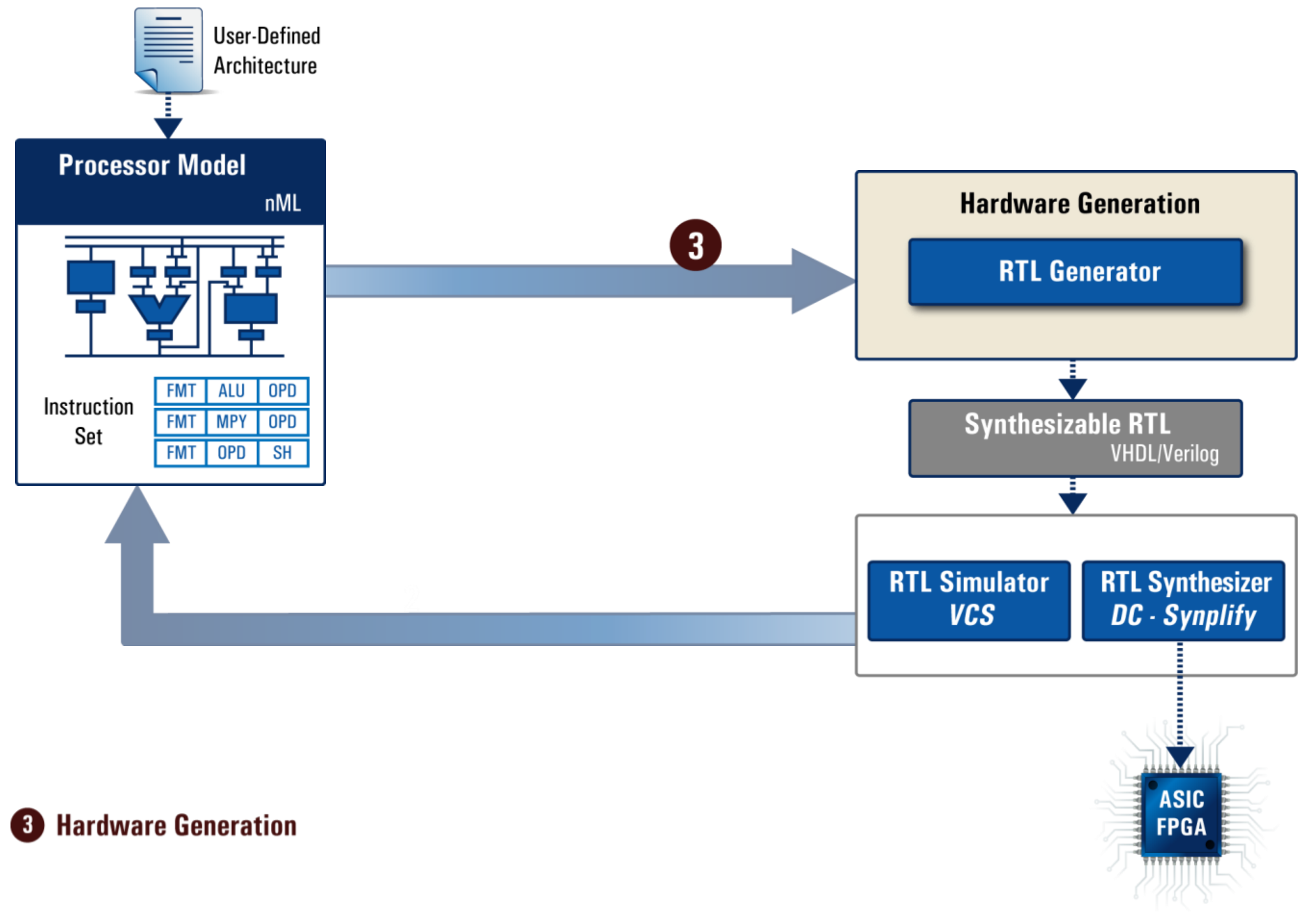


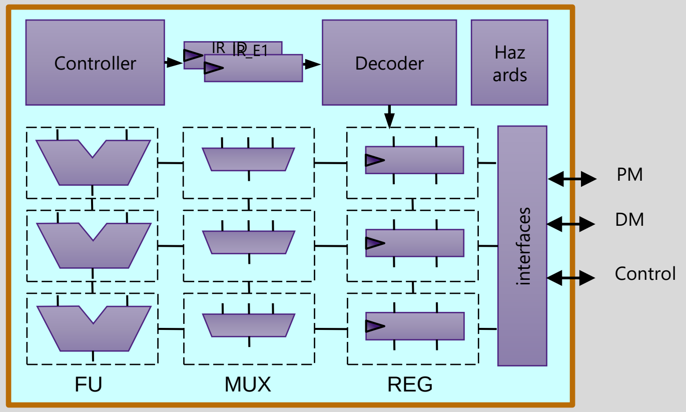
  - Register allocation for distributed register files



ASIP Designer

Production Ready RTL Generation

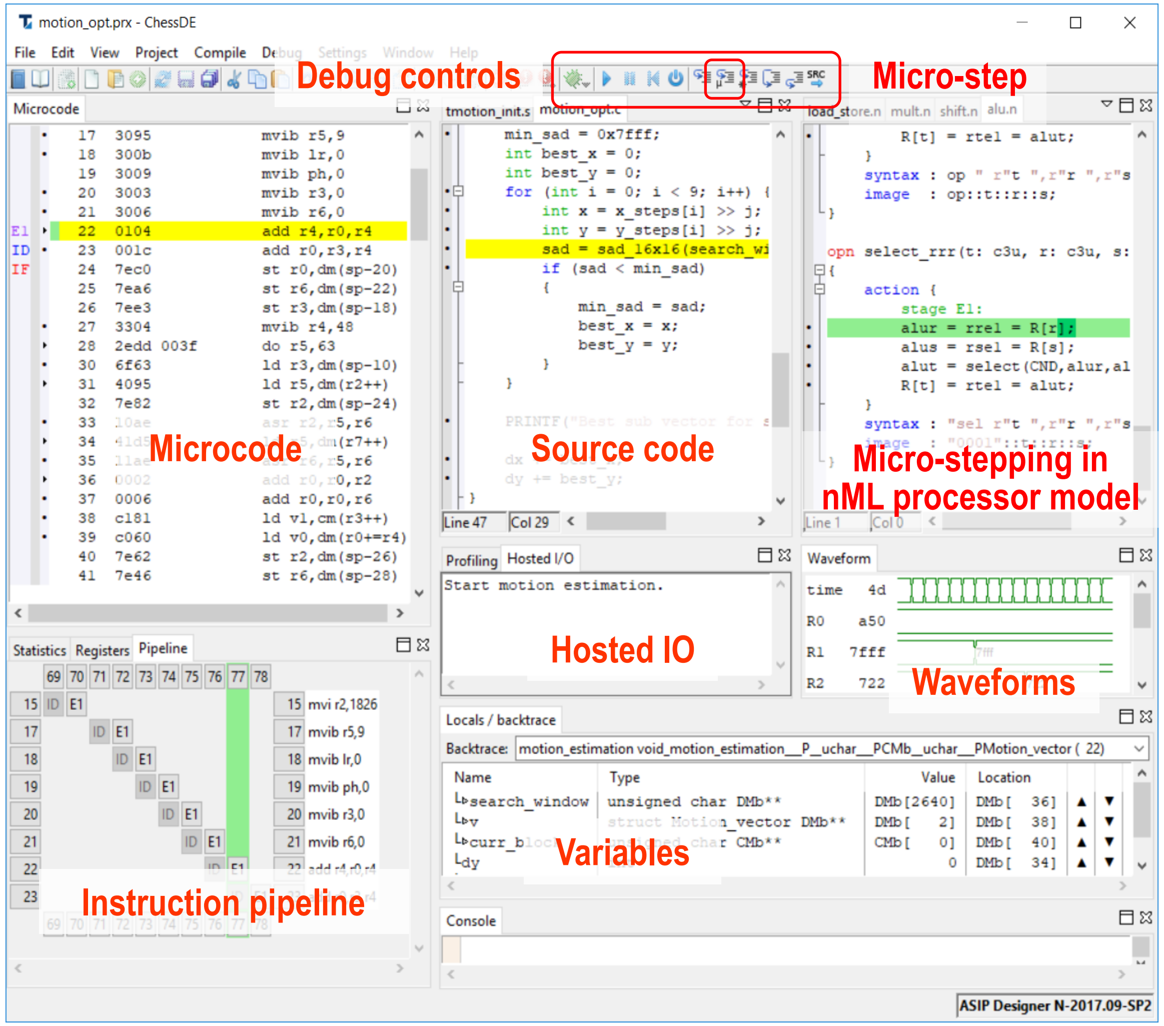




- Converts nML processor model into synthesizable Verilog or VHDL
- Generated RTL code is netlist of modules: functional units, register files, muxes, controller, …
- Low power features
  - Operand isolation
  - Clock gating
- Automatic synthesis script generation
- Use real synthesis and P&R data to measure gate count, timing closure

Integrated Debugging Flow

ChessDE's Graphical Debugger Perspective



```
E1  22  0104            add r4,r0,r4
ID  23  001c            add r0,r3,r4
IF  24  7ec0            st r0,dm(sp-20)
    17  3095            mvib r5,9
    18  300b            mvib lr,0
    19  3009            mvib ph,0
    20  3003            mvib r3,0
    21  3006            mvib r6,0
    25  7ea6            st r6,dm(sp-22)
    26  7ee3            st r3,dm(sp-18)
    27  3304            mvib r4,48
    28  2edd 003f       do r5,63
    30  6f63            ld r3,dm(sp-10)
    31  4095            ld r5,dm(r2++)
    32  7e82            st r2,dm(sp-24)
    33  10ae            asr r2,r5,r6
    34  41d5            ld r5,dm(r7++)
    35  11ae            asr r6,r5,r6
    36  0002            add r0,r0,r2
    37  0006            add r0,r0,r6
    38  c181            ld v1,cm(r3++)
    39  c060            ld v0,dm(r0+=r4)
    40  7e62            st r2,dm(sp-26)
    41  7e46            st r6,dm(sp-28)
```

tmotion_init.s motion_opt.c

```
    min_sad = 0x7fff;
    int best_x = 0;
    int best_y = 0;
    for (int i = 0; i < 9; i++) {
        int x = x_steps[i] >> j;
        int y = y_steps[i] >> j;
        sad = sad_16x16(search_wi
        if (sad < min_sad)
        {
            min_sad = sad;
            best_x = x;
            best_y = y;
        }
    }

    PRINTF("Best sub vector for s

    dx += best_x;
    dy += best_y;
}
```

Line 47 Col 29

load_store.n mult.n shift.n alu.n

```
        R[t] = rtel = alut;
    }
    syntax : op " r"t ",r"r ",r"s
    image  : op::t::r::s;
}

opn select_rrr(t: c3u, r: c3u, s:
{
    action {
        stage E1:
        alur = rrel = R[r];
        alus = rsel = R[s];
        alut = select(CND,alur,al
        R[t] = rtel = alut;
    }
    syntax : "sel r"t ",r"r ",r"s
    image  : "0001"::t::r::s;
}
```

Line 1 Col 0

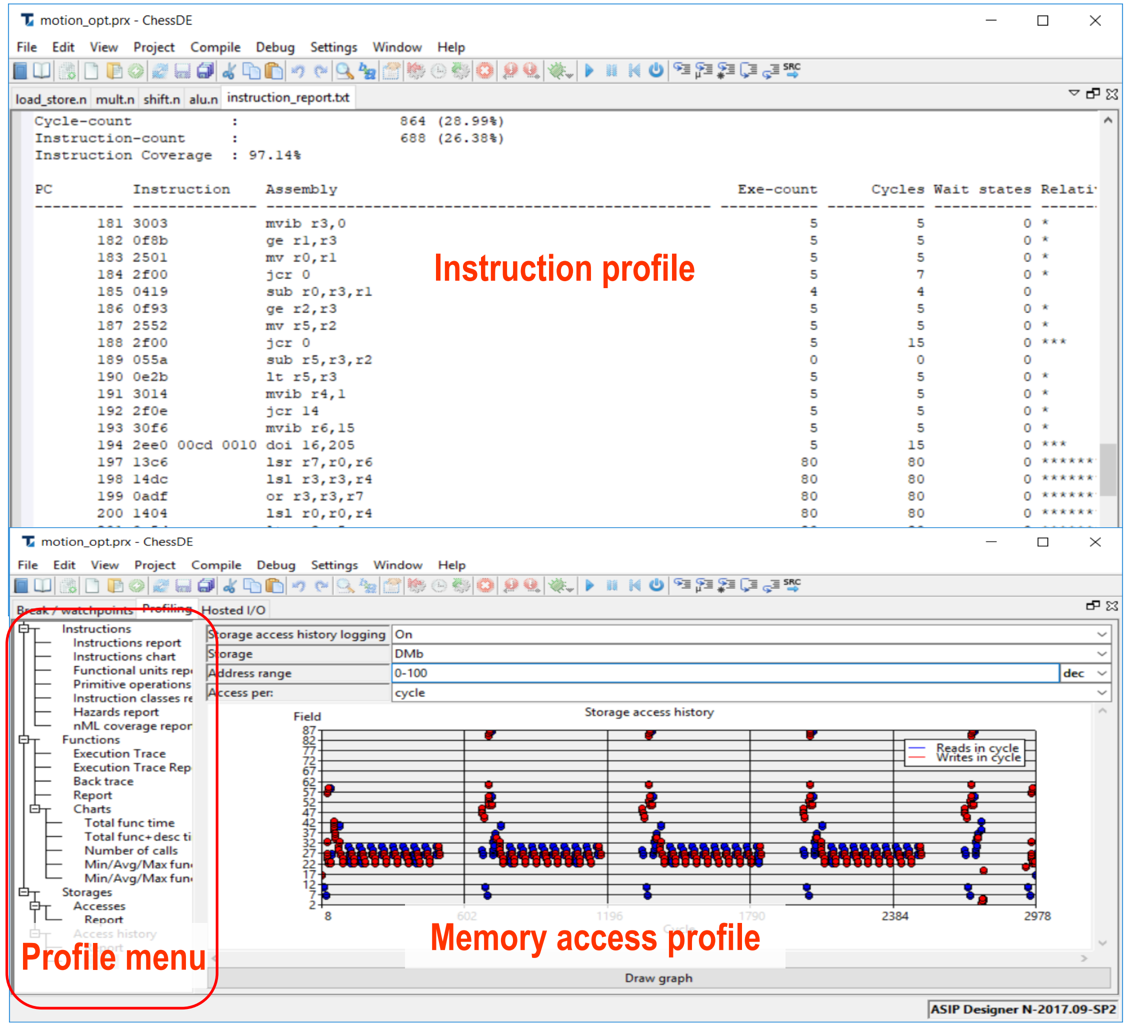
```
Start motion estimation.
```

```
time  4d
R0    a50
R1    7fff
R2    722
```

Backtrace: motion_estimation void_motion_estimation__P__uchar__PCMb__uchar__PMotion_vector (22)

Name	Type	Value	Location
search_window	unsigned char DMb**	DMb[2640]	DMb[36]
v	struct Motion_vector DMb**	DMb[2]	DMb[38]
curr_block	unsigned char CMb**	CMb[0]	DMb[40]
dy		0	DMb[34]

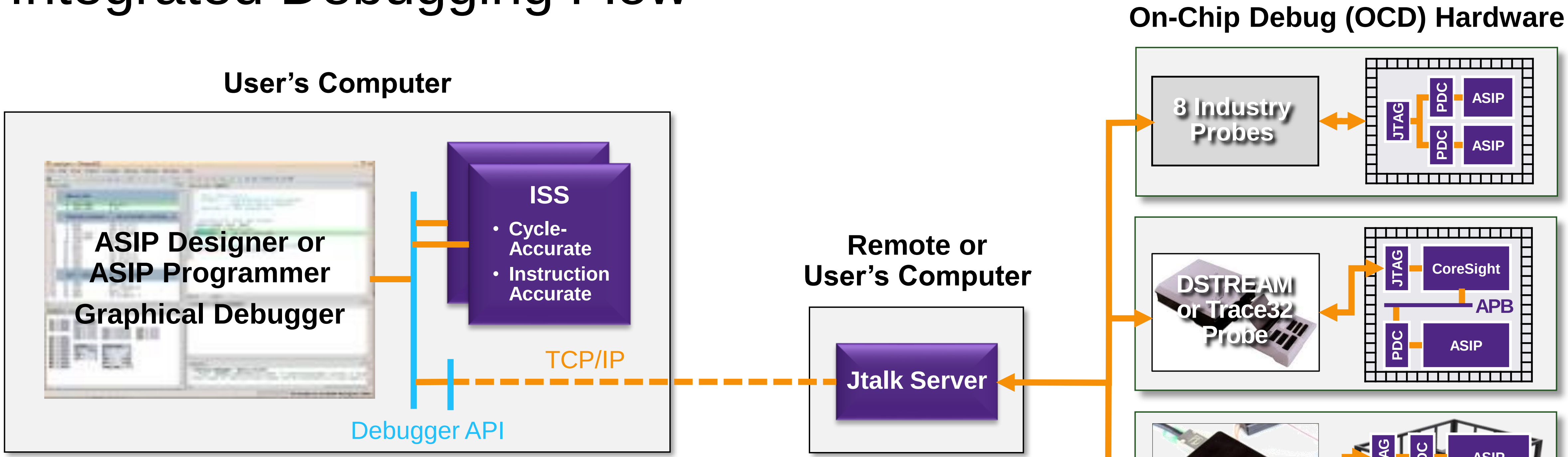
```
69 70 71 72 73 74 75 76 77 78
15 ID E1                 15 mvi r2,1826
17    ID E1              17 mvib r5,9
18       ID E1           18 mvib lr,0
19          ID E1        19 mvib ph,0
20             ID E1     20 mvib r3,0
21                ID E1  21 mvib r6,0
22                   ID E1  22 add r4,r0,r4
23
```



```
Cycle-count            :              864 (28.99%)
Instruction-count      :              688 (26.38%)
Instruction Coverage   : 97.14%

PC         Instruction    Assembly                        Exe-count      Cycles Wait states Relati
---------- ------------- ------------------------------- ----------- ----------- ----------- ------
       181 3003           mvib r3,0                               5           5           0 *
       182 0f8b           ge r1,r3                                5           5           0 *
       183 2501           mv r0,r1                                5           5           0 *
       184 2f00           jcr 0                                   5           7           0 *
       185 0419           sub r0,r3,r1                            4           4           0
       186 0f93           ge r2,r3                                5           5           0 *
       187 2552           mv r5,r2                                5           5           0 *
       188 2f00           jcr 0                                   5          15           0 ***
       189 055a           sub r5,r3,r2                            0           0           0
       190 0e2b           lt r5,r3                                5           5           0 *
       191 3014           mvib r4,1                               5           5           0 *
       192 2f0e           jcr 14                                  5           5           0 *
       193 30f6           mvib r6,15                              5           5           0 *
       194 2ee0 00cd 0010 doi 16,205                              5          15           0 ***
       197 13c6           lsr r7,r0,r6                           80          80           0 ******
       198 14dc           lsl r3,r3,r4                           80          80           0 ******
       199 0adf           or r3,r3,r7                            80          80           0 ******
       200 1404           lsl r0,r0,r4                           80          80           0 ******
```

Storage access history logging	On
Storage	DMb
Address range	0-100 dec
Access per:	cycle

Integrated Debugging Flow



Graphical Debugger	Debug Flow	No. Cores
ChessDE	ISS, OCD (single-core)	Single
ChessMP	ISS, OCD (multi-core)	Multi
Eclipse	ISS, OCD	Single
Virtualizer	Virtual prototyping (SystemC)	Multi
Verdi	ISS (tracing info from RTL)	Single

- Jtalk server
  - Built-in drivers for 11 industry OCD probes
  - Dynamic linking with customer-defined OCD probe driver
- ASIP Designer generates RTL of:
  - ASIP, PDC (processor debug controller), JTAG, APB bus

Trv (RISC-V ISA) Models

Overview

Integer models

- Supported ISA: RV64IM, RV32IM
  - Integer instructions
  - Multiply instructions
- Micro architecture
  - Protected pipeline with 3 or 5 stages
  - Hardware multiplier
  - Iterative divider
- Optional extensions: Trv<mm>p<n>x
  - Compressed instructions
  - Zero overhead hardware loops
  - Load/stores with post-modify address modes

	32-bit datapath	**64-bit datapath**
3-stage pipeline	Trv32p3 Trv32p3x	Trv64p3 Trv64p3x
5-stage pipeline	Trv32p5 Trv32p5x	Trv64p5 Trv64p5x

Floating point models

- Supported ISA: RV64IMF, RV32IMF
  - Integer instructions
  - Multiply instructions
  - Float (single precision) instructions
- Micro architecture
  - DesignWare based implementation FPUs
  - Iterative divider and sqrt units
- Optional extensions: Trv<mm>p<n>fx
  - Zero overhead hardware loops
  - Load/stores with post-modify address modes

	32-bit datapath
3-stage pipeline	Trv32p3f Trv32p3fx
5-stage pipeline	Trv32p5f Trv32p5fx

SDX

- Simple Data Path Extension Mechanism
  - FFT
  - SHA256
  - CNN

Tmoby

- SIMD/VLIW MobileNet accelerator

Future work

- Model improvements
- Privileged Architecture
- RISC-V Vector ISA

Trv (RISC-V ISA) Models

Integer Variants

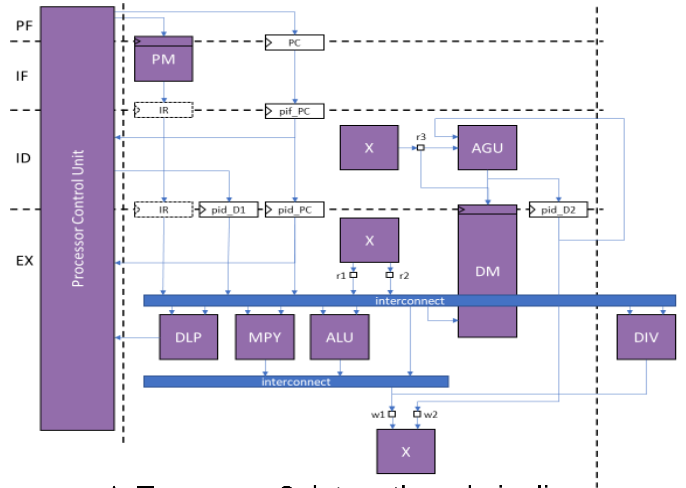
Context

- Optimized processor models of RISC-V ISA
- Intended as starting point for ISA extension (ASIP design)

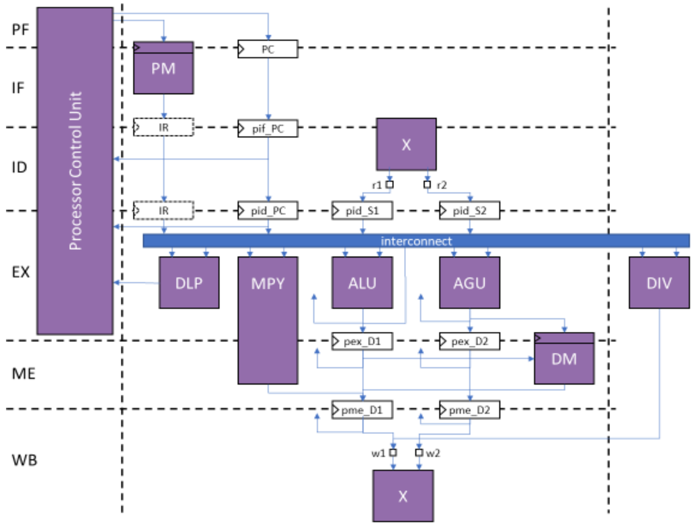
	32-bit datapath	64-bit datapath
3-stage pipeline	Trv32p3 Trv32p3x	Trv64p3 Trv64p3x
5-stage pipeline	Trv32p5 Trv32p5x	Trv64p5 Trv64p5x

- Supported ISA: RV64IM, RV32IM
  - Integer instructions
  - Multiply instructions
- Optional extensions: Trv<mm>p<n>x
  - Compressed instructions
  - Zero overhead hardware loops
  - Load/stores with post-modify address modes

- Micro architecture features
  - Five stage pipeline: IF, ID, EX, ME, WB
  - Three stage pipeline: IF, ID, EX
  - Protected pipeline
    - Register bypasses when possible
    - Stall (bubble) when bypass not possible
  - Hardware multiplier (64x64→64 bit, 32x32→32bit)
  - Iterative divider



▲ Trx<mm>p3 datapath and pipeline



▲ Trx<mm>p5 datapath and pipeline

Trv (RISC-V ISA) Models

SDX: Simple Datapath eXtensions

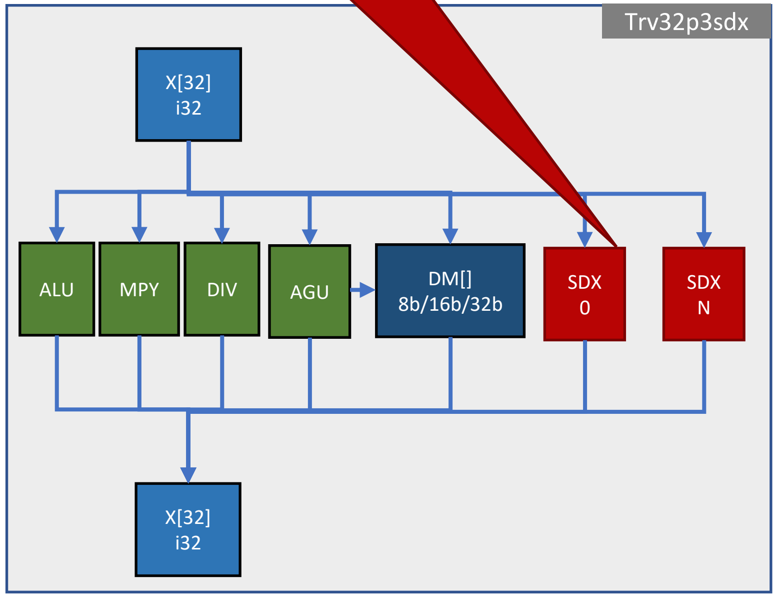
- SDX is a mechanism to add simple extension instructions to Trv (RISC-V)
  - nML model of Trv32p3 with predefined **stubs** for extension instructions
  - User codes the **behavior** of the stubs in PDG (bit-accurate C code)
  - Compiler intrinsics that target the extension instructions are provided (and can be modified)
- Benefits
  - No (deep) nML knowledge required: extensions can be created by SW engineers
  - Fast exploration of extensions with compiler-in-the-loop & synthesis-in-the-loop
- Extension encoded in RISC-V custom-2 space
- Operand options
  - 3-register (32 and 64-bit)
  - Accumulate
  - Additional single register inputs and outputs, …

Behavioral model (PDG)

```
w32 sdx0(w32 a, w32 b)
{
  w32 r;
  r[15:0] = a[15:0] + b[15:0];
  r[31:16] = a[31:16] + b[31:16];
  return r;
}
```

Compiler intrinsics

```
int sdx0(int,int);
int add2(int,int);
```



Trv (RISC-V ISA) Models

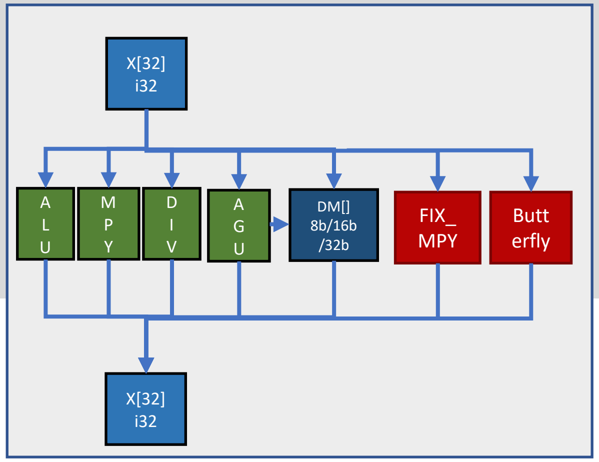
SDX Examples Provided With ASIP Designer

FFT

SDX instructions accelerating
- Complex fixed-point multiplication & scaling
  sdx1 *rd,rs1,rs2*
- ABS(x) function: ***sdx2*** *rd, rs1, rs2(x0)*
- FFT Butterfly : ***sdx5*** *rd,rs1,rs1*

Specialization:
- Fractional data types
- Complex numbers (16bit/16bit -> 32bit register)

Speedup: 280% Area increase: 31%



SHA256

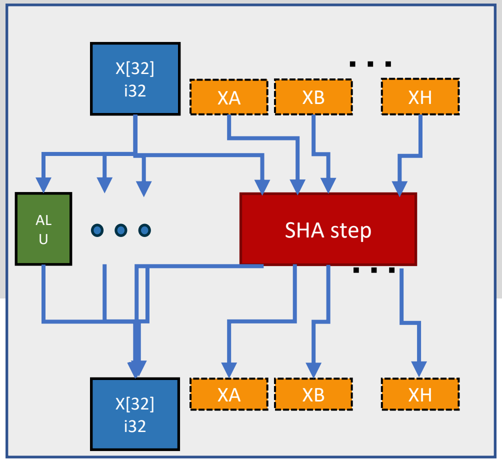
Computes a hash of message W using bitwise AND, OR, XOR operations, shift operations and additions

Custom data path is ideal to implement the complex hash function in one instruction

Additional state of the hash functions (8 state variables) require an SDX variant that supports **8 additional register reads and writes**

sdx7 rd, rs1, rs2, x24,x25,…, x32

Speedup: 270% Area increase: 16%



Keyword Spotting

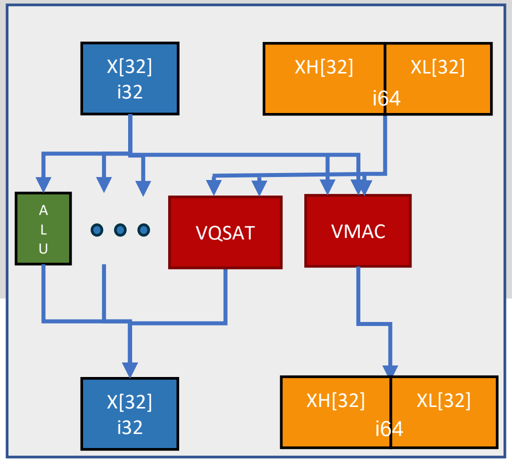
Based on small sized Neural Network (3.3M MACs)

SDX architecture feature: **packed SIMD**
32-bit register contains vector of 4x 8-bit values 🟩🟥🟦⬜

Use of register pairs, enabling 64-bit access

```
sdx4a_dr dd,rs1,rs2,mode // vmac
sdx0_dd  rd, ds1,ds2     // vqsat
```

Speedup: 1160% Area increase: 16%



Trv (RISC-V ISA) Models

Tmoby: Accelerator Tuned for MobileNet v3 Execution with Medium Throughput

Context

- Growing interest in ASIPs for AI applications
- Customer requests to provide educational models, demonstrating how ASIPs can accelerate AI functions

Architecture

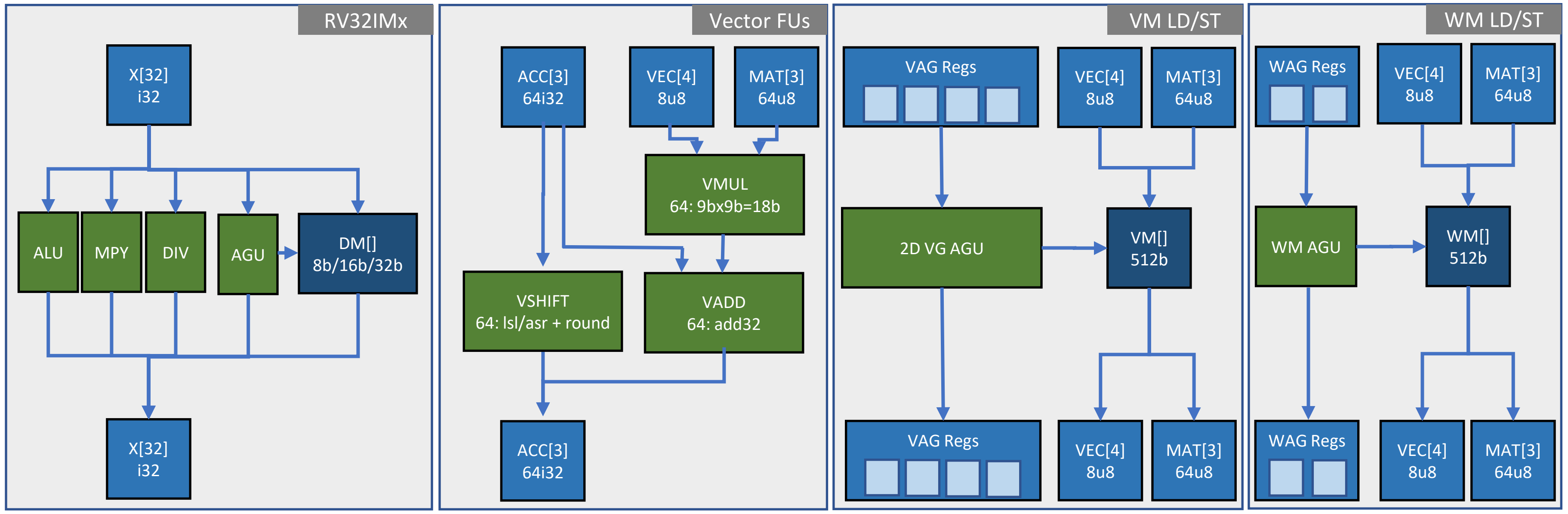
- 4-way ILP
- Trv32p3-based scalar datapath
- Vector datapath
  - SIMD64
  - MAC 8x8→32
- Memories
  - VM: features
  - WM: weights
  - Vector addressing

Application

- Main function
  - Implementation of neural-network graph
  - Memory copies
  - Kernels
- Kernels
  - CONV_2D (pointwise)
  - DEPTHWISE_CONV_2D
  - ADD
  - AVERAGE_POOL_2D
  - SOFTMAX

Functionality 2020.09

- Tmoby: educational example model
- Application: MobileNet V3
  - Based on TensorFlow code
  - Converted to C code with Tmoby-specific vector intrinsics
- Starting from RISC-V ISA
- 360x cycle cnt. decrease



RVV Proof of Concept Models

- Goal: demonstrate and document to potential users how RISC-V Vector instructions can be modelled in ASIP Designer
  - Modelling of basic RVV concepts
    - Opcode polymorphism (residual control based on SEW register)
    - Register grouping (residual control based on LMUL register)
    - Whole vector instructions for moves and spilling (needed for C code compilation)
  - Dynamic multi-issue model
    - How can dynamic multi-issuing be modeled in ASIP Designer
  - Pure ASIP VLIW style model
    - Support RVV C API using static instruction level parallelism
- Notice: these models are not feature complete nor RTL-optimized
  - Only a few instructions are supported: vsetvli, vadd, vle, vse, vmv, vmvNr, vlNr, vsNr
  - Shallow FDE pipeline with one execute stage

- Problem statement
  - RVV code is a mix of scalar and vector opcodes. When executed in a scalar way the vector units are used in an inefficient way

```
saxpy :
   vsetvli a4, a0, e32, m8
   vlw.v v0, (a1)
   sub a0, a0, a4
   slli a4, a4, 2
   add a1, a1, a4
   vlw.v v8, (a2)
   vfmacc.vf v8, fa0, v0
   vsw.v v8, (a2)
   add a2, a2, a4
   bnez a0, saxpy
```

  - Only 4 out of 10 instructions are vector instructions

RVV Dynamic multi-issue versus VLIW

ASIP Designer allows you to explore both options and their variants

- Option 1: dynamic multi-issue
  - Three-way multi-issue model

Scalar opcodes	Vector compute	Vector load/store

  - Issue logic converts scalar code into parallel code
- Proof of concept model also implements register grouping using serial execution and chaining
  - Example, assuming LMUL = 4

```
vadd v8,v0,v4
vst v8,(a0)
```

```
scalar 1 | vadd v8, v0,v4
scalar 2 | vadd v9, v1,v5 | vst v8,(a0)
scalar 3 | vadd v10,v2,v6 | vst v9,(a0+L)
scalar 4 | vadd v11,v3,v7 | vst v10,(a0+2L)
scalar 5 |                | vst v11,(a0+3L)
```

- RVV binary compatible model

- Option 2: ASIP VLIW Architecture (static multi-issue)
  - Support the RISC-V Vector C API by hosting the RISC-V vector instructions in a VLIW structure
- More architecture freedom
  - Three slot VLIW
  - Zero overhead loops
  - Post increment addressing modes
  - Dual width vector loads
  - Register grouping: wide vectors are converted to short vectors in the compiler front end
- Better leverages ASIP Designer's capabilities
  - Complex dynamic issue logic is replaced by straightforward issue logic
  - Leverage powerful and unique compiler technology to optimize code at compile time
- C code compatibility model (support the RVV C API)

NSITEXE Develops Multiple RISC-V Based Custom Processors for Automotive Applications in Half the Time with Synopsys ASIP Designer

Rapid Architectural Exploration and Automatic Generation of SDKs Accelerates Design Time

NSITEXE Successfully Develops Multiple Custom Processors for Automotive Applications in Half the Time with Synopsys ASIP Designer Tool

ASIP Designer Tool Provides Rapid Architectural Exploration and Automatic Generation of Software Development Kits to Accelerate Design Time

MOUNTAIN VIEW, Calif., Sept. 16, 2020 /PRNewswire/ --

- NSITEXE used ASIP Designer to develop five specialized custom processors for compute-intensive signal processing functions
- Automatic generation of software development kit enabled NSITEXE to complete the design with lir
- R ab

Synops Corpor Design proces was ab develo

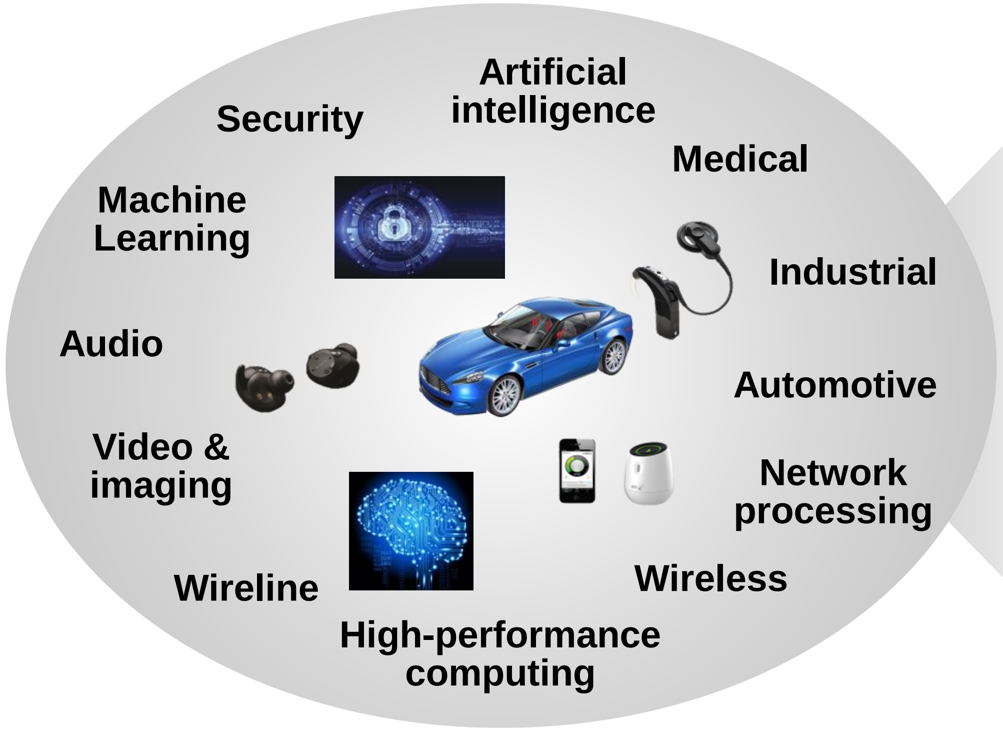
“Synopsys' ASIP Designer Tool provided us with ready-to-use processor examples that could be extended to implement our custom ISA, allowing us to rapidly develop our custom vector extensions that satisfied our functionality and performance requirements. ASIP Designer enabled us to develop the data flow processor model in record time and deliver it to the customer on schedule.”

Sadahiro Kimura,
Manager, Semiconductor IP R&D Unit,
Advanced Technology Development Section at NSITEXE, Inc.

- NSITEXE used ASIP Designer to develop five specialized custom processors for compute-intensive signal processing functions
- Automatic generation of software development kit enabled NSITEXE to complete the design with limited resources and on a tight schedule
- Rapid setup of a virtual prototype of NSITEXE's multicore DFP enabled by the ASIP Designer’s ability to export the processor's simulation model with SystemC interfaces

Synopsys' ASIP Tools

Broadest Market Adoption – Industry Proven



Widespread Deployment Across A Wide Range Of Applications

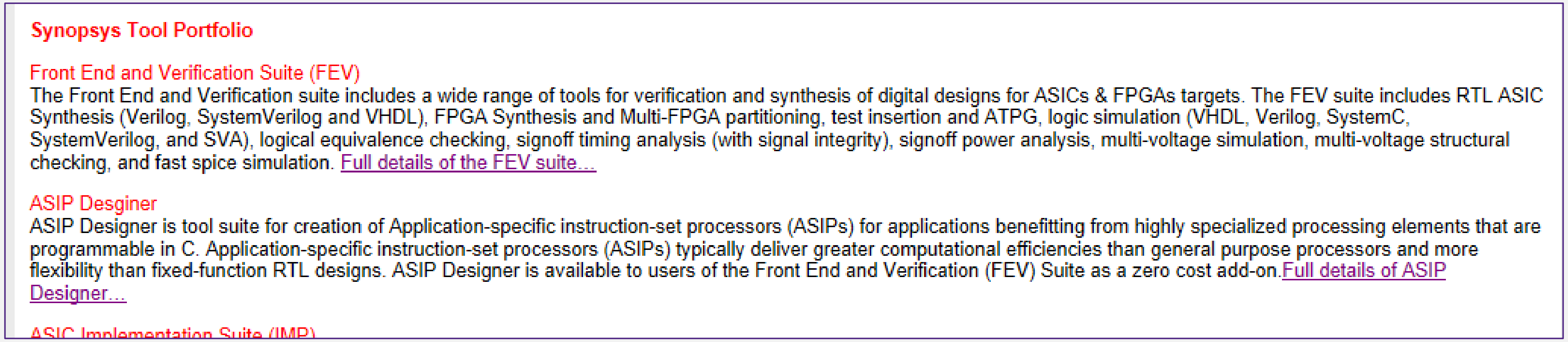
- Estimate more than 300 unique SoC products with ASIPs in the market today
- Shown are publicly announced customers only

Synopsys University Program

Synopsys University Program – Europe Region

Managed by Europractice

- Since Jan’ 2019: ASIP Designer available free-of-charge on top of Synopsys FEV package



Synopsys Tool Portfolio

Front End and Verification Suite (FEV)
The Front End and Verification suite includes a wide range of tools for verification and synthesis of digital designs for ASICs & FPGAs targets. The FEV suite includes RTL ASIC Synthesis (Verilog, SystemVerilog and VHDL), FPGA Synthesis and Multi-FPGA partitioning, test insertion and ATPG, logic simulation (VHDL, Verilog, SystemC, SystemVerilog, and SVA), logical equivalence checking, signoff timing analysis (with signal integrity), signoff power analysis, multi-voltage simulation, multi-voltage structural checking, and fast spice simulation. Full details of the FEV suite...

ASIP Desginer
ASIP Designer is tool suite for creation of Application-specific instruction-set processors (ASIPs) for applications benefitting from highly specialized processing elements that are programmable in C. Application-specific instruction-set processors (ASIPs) typically deliver greater computational efficiencies than general purpose processors and more flexibility than fixed-function RTL designs. ASIP Designer is available to users of the Front End and Verification (FEV) Suite as a zero cost add-on.Full details of ASIP Designer...

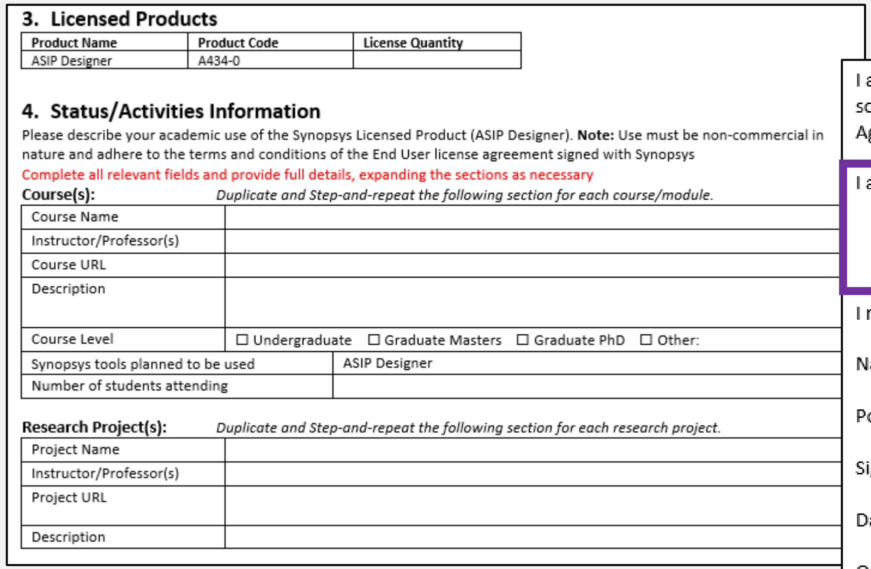
ASIC Implementation Suite (IMP)

  - On request only: approval required, additional document to be signed

Synopsys University Program

Other regions

- The Americas
  - ASIP Designer available free-of-charge on top of Synopsys University Package
  - On request only: approval required, additional document to be signed



3. Licensed Products

Product Name	Product Code	License Quantity
ASIP Designer	A434-0	

4. Status/Activities Information

Please describe your academic use of the Synopsys Licensed Product (ASIP Designer). **Note:** Use must be non-commercial in nature and adhere to the terms and conditions of the End User license agreement signed with Synopsys

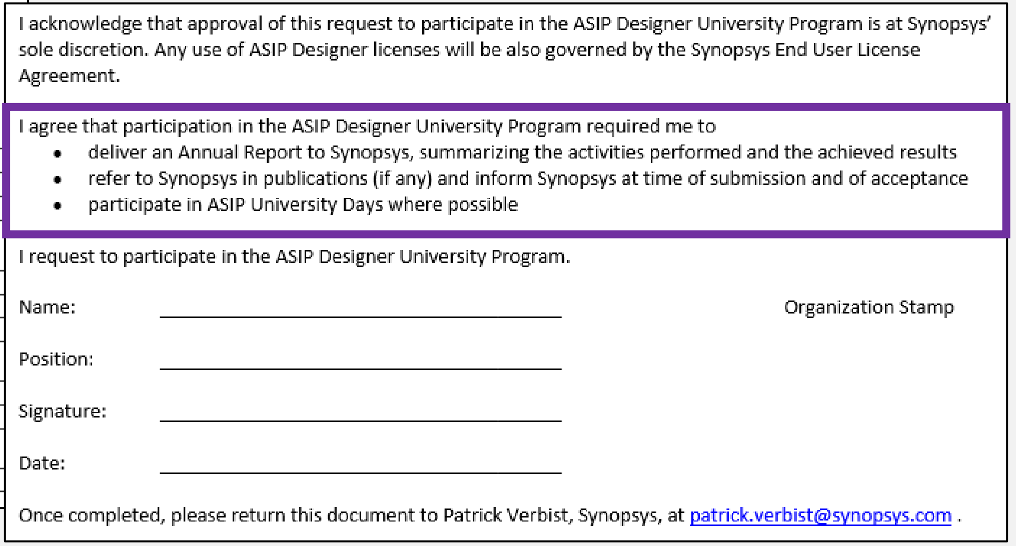
Complete all relevant fields and provide full details, expanding the sections as necessary

Course(s): *Duplicate and Step-and-repeat the following section for each course/module.*

Course Name	
Instructor/Professor(s)	
Course URL	
Description	
Course Level	☐ Undergraduate ☐ Graduate Masters ☐ Graduate PhD ☐ Other:
Synopsys tools planned to be used	ASIP Designer
Number of students attending	

Research Project(s): *Duplicate and Step-and-repeat the following section for each research project.*

Project Name	
Instructor/Professor(s)	
Project URL	
Description	



I acknowledge that approval of this request to participate in the ASIP Designer University Program is at Synopsys' sole discretion. Any use of ASIP Designer licenses will be also governed by the Synopsys End User License Agreement.

I agree that participation in the ASIP Designer University Program required me to
- deliver an Annual Report to Synopsys, summarizing the activities performed and the achieved results
- refer to Synopsys in publications (if any) and inform Synopsys at time of submission and of acceptance
- participate in ASIP University Days where possible

I request to participate in the ASIP Designer University Program.

Name: ______________________ Organization Stamp

Position: ______________________

Signature: ______________________

Date: ______________________

Once completed, please return this document to Patrick Verbist, Synopsys, at patrick.verbist@synopsys.com .

- Other regions
  - Please contact patrick.verbist@synopsys.com

Silicon to Software™

Thank You

Event survey link: https://www.surveymonkey.com/r/ASIP-univ-day2021