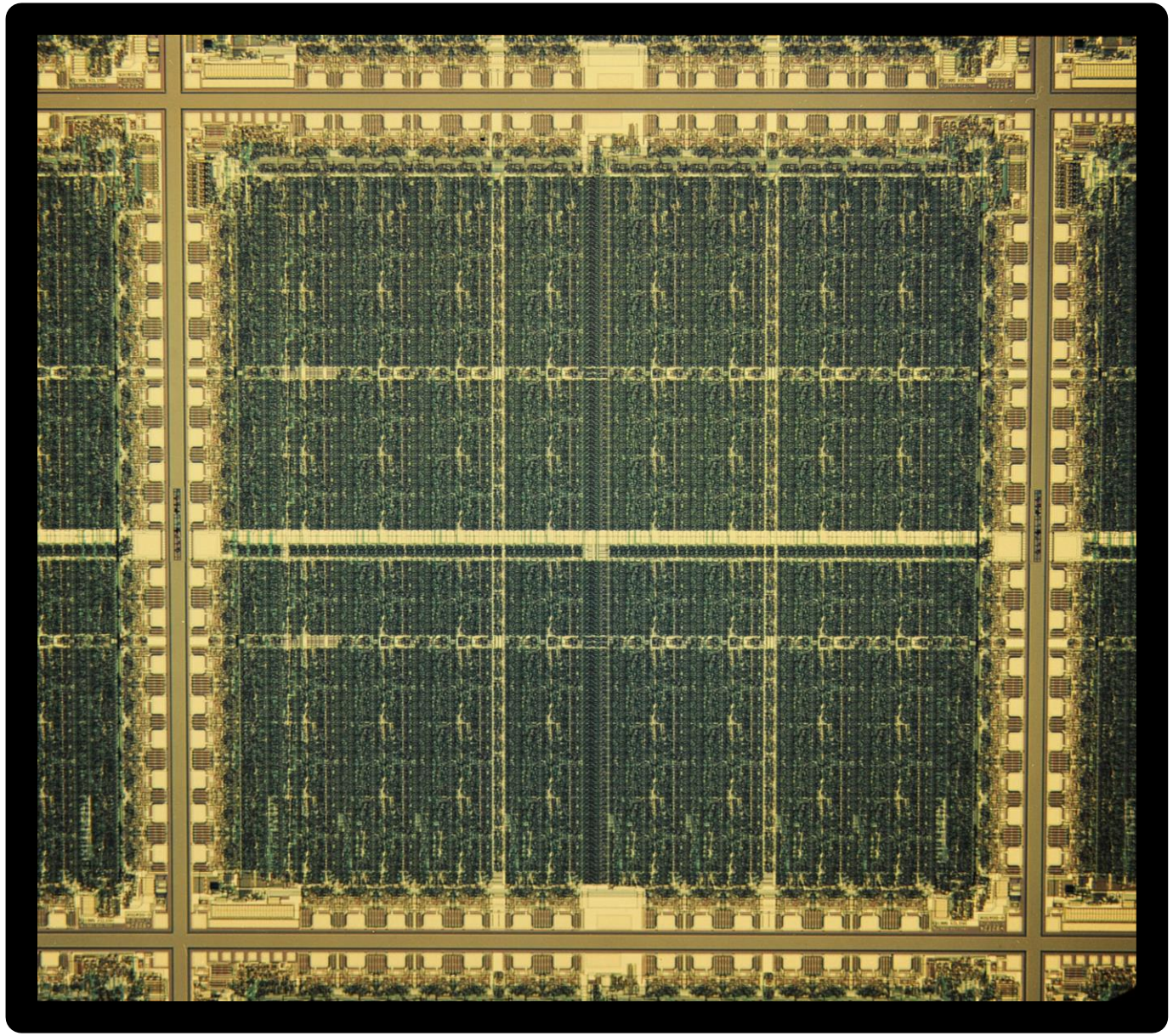
XILINX
ALL PROGRAMMABLE™

FPGA & high-level programming tools

Ronan Keryell, Xilinx Research Labs
ANL REFORM 2016/01/21

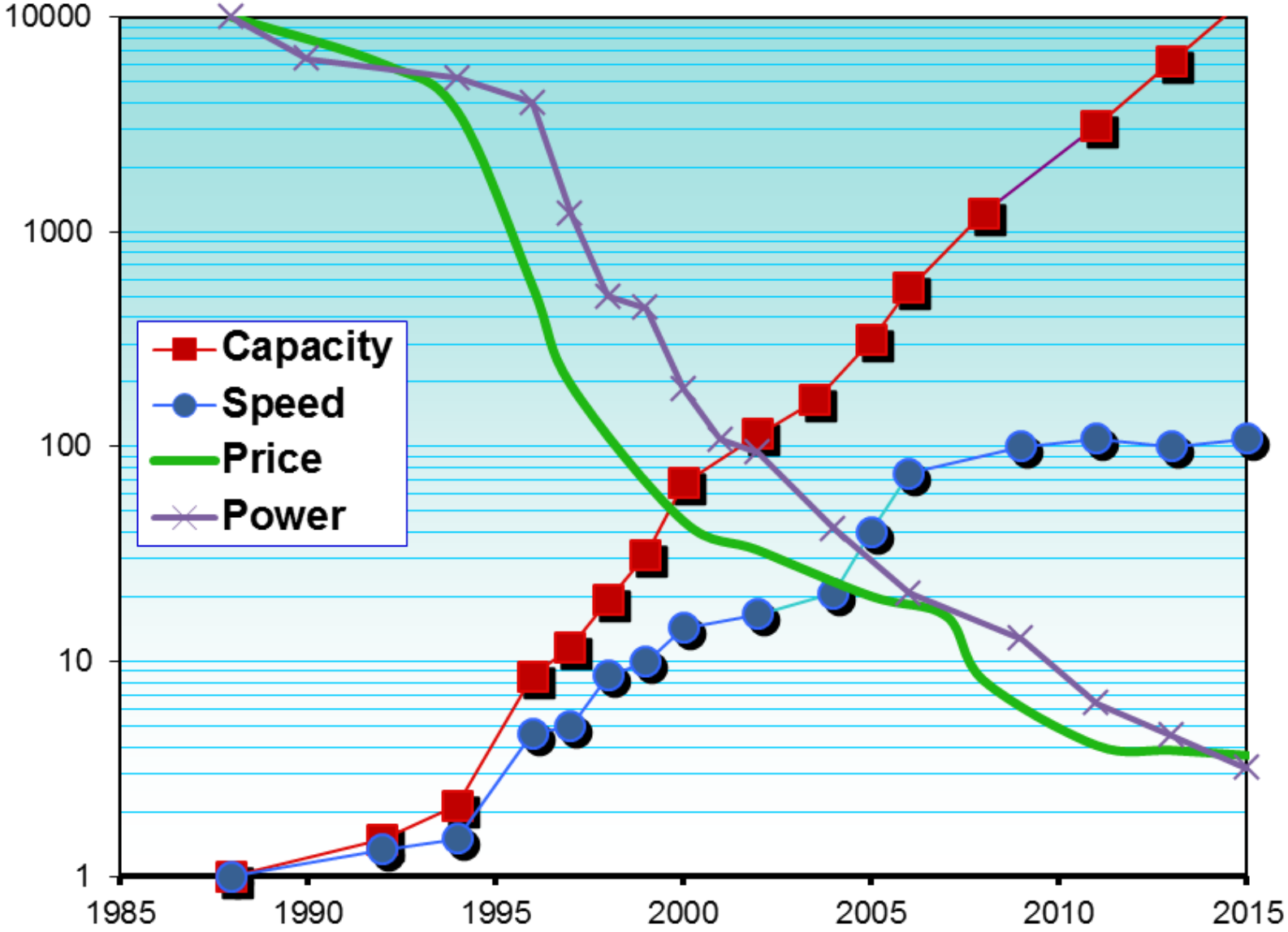
Once upon a time the XC2064…



1985: the First FPGA

- **64 flip-flops**
- **128 3-LUT**
- **58 I/O pins**
- **18 MHz (toggle)**
- **2 µm 2LM**

Since then…



- **10,000x more logic…**
  - Plus embedded IP
    - Memory
    - Microprocessor
    - DSP
    - Gigabit Serial I/O
- **100x faster**
- **5,000x lower power/gate**
- **10,000x lower cost/gate**

Three Ages of the FPGAs: Trimberger, S, Proceedings of the IEEE | Vol. 103, No. 3, March 2015

Next generation challenges

- Power
- Performance
- Cost drivers
- Power management
- 64bit processing
- Real-time processing
- Video and graphics processing
- Pervasive safety and security
- Higher levels of processor-fabric integration

Introducing the Zynq UltraScale+ MPSoC

ARM Cortex A53 & R5

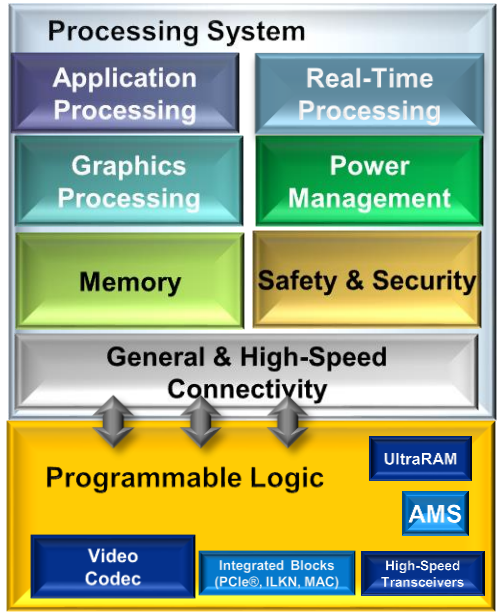
- Application processing subsystem
- Real Time processing subsystem
- Memory subsystem for max performance

IO, Video, Graphics

- Next-generation coherent interconnect
- High-speed I/Os (PCIE,USB3,SATA,GbE)
- Graphics and Video Processing Engines

Advanced Power Mgmt

- Fine-grained power reduction
- System-level software & run time opt



Security & Safety

- Enhanced Authentication, Encryption, Antitamper and trust
- Safety with industry standards support

Fabric Acceleration

- UltraScale+ fabric with time borrow
- FinFET performance and power
- HD UlltraRAM, and enhanced DSP

XCVRs & Protocols

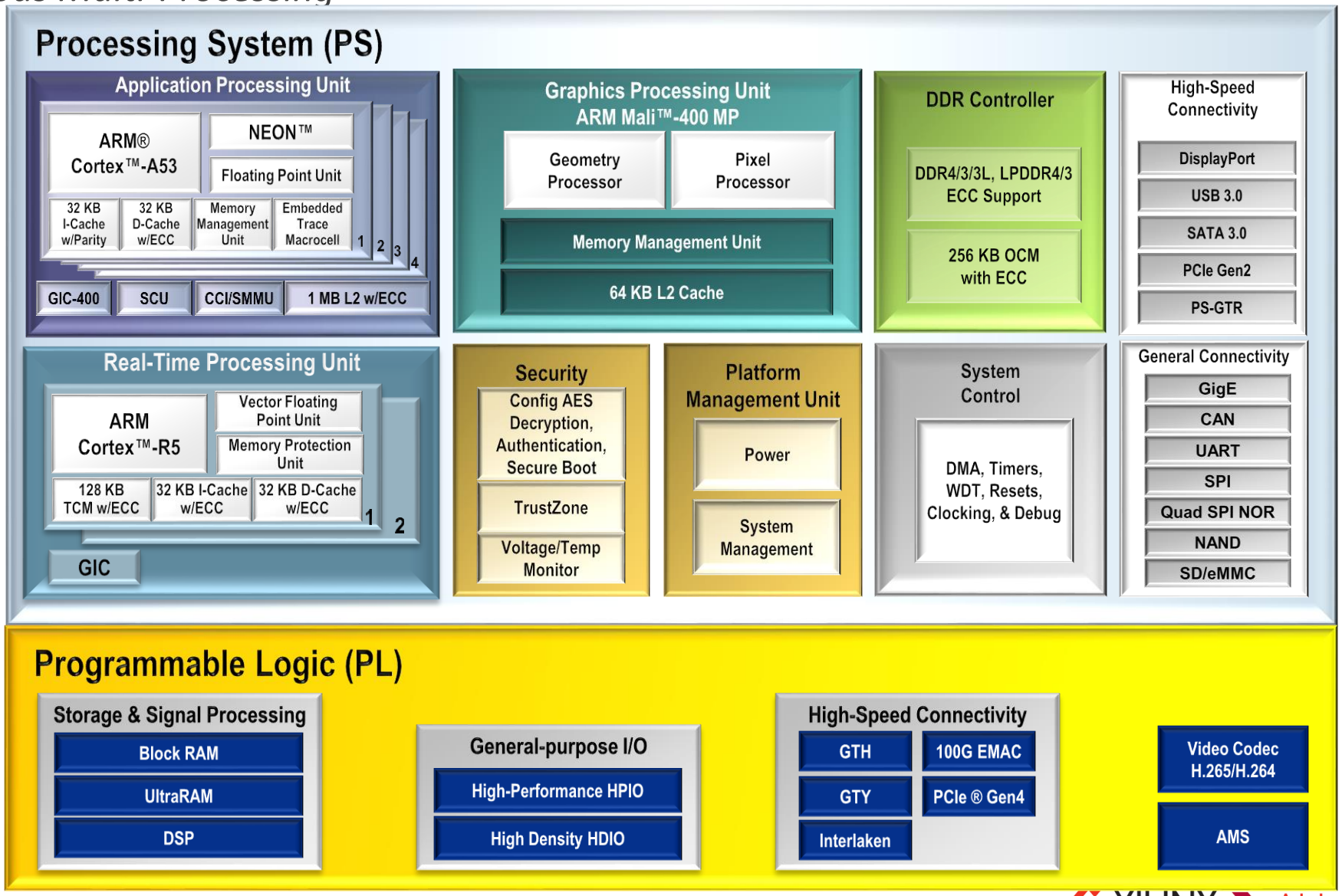
- Power efficient, 32Gbps
- 100G Ethernet and 150G Interlaken
- PCIe Gen3 & Gen4

Software & Tools

Run Time (Xilinx)	Run Time (Ecosystem)	Tools
- Linux (64b)	- FreeRTOS	- Xilinx SDK
- Hypervisor	- Micrium	- Vivado®
- OpenAMP	- WindRiver & More	- SDx environments

Zynq UltraScale+ MPSoC Overview

Heterogeneous Multi-Processing



Introducing the Zynq UltraScale+ MPSoC

ARM Cortex A53 & R5

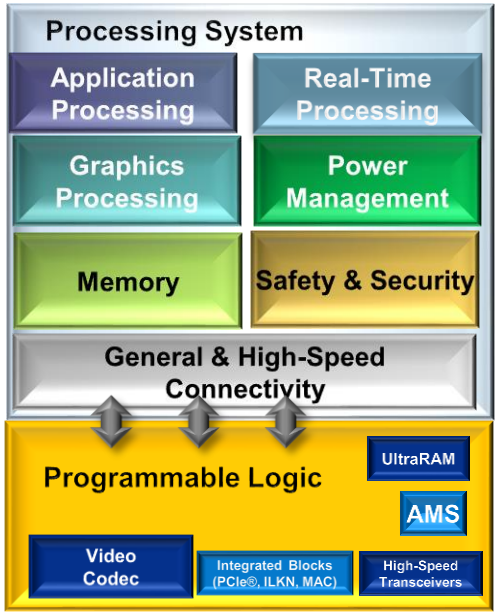
- Application processing subsystem
- Real Time processing subsystem
- Memory subsystem for max performance

IO, Video, Graphics

- Next-generation coherent interconnect
- High-speed I/Os (PCIE,USB3,SATA,GbE)
- Graphics and Video Processing Engines

Advanced Power Mgmt

- Fine-grained power reduction
- System-level software & run time opt



Security & Safety

- Enhanced Authentication, Encryption, Antitamper and trust
- Safety with industry standards support

Fabric Acceleration

- UltraScale+ fabric with time borrow
- FinFET performance and power
- HD UlltraRAM, and enhanced DSP

XCVRs & Protocols

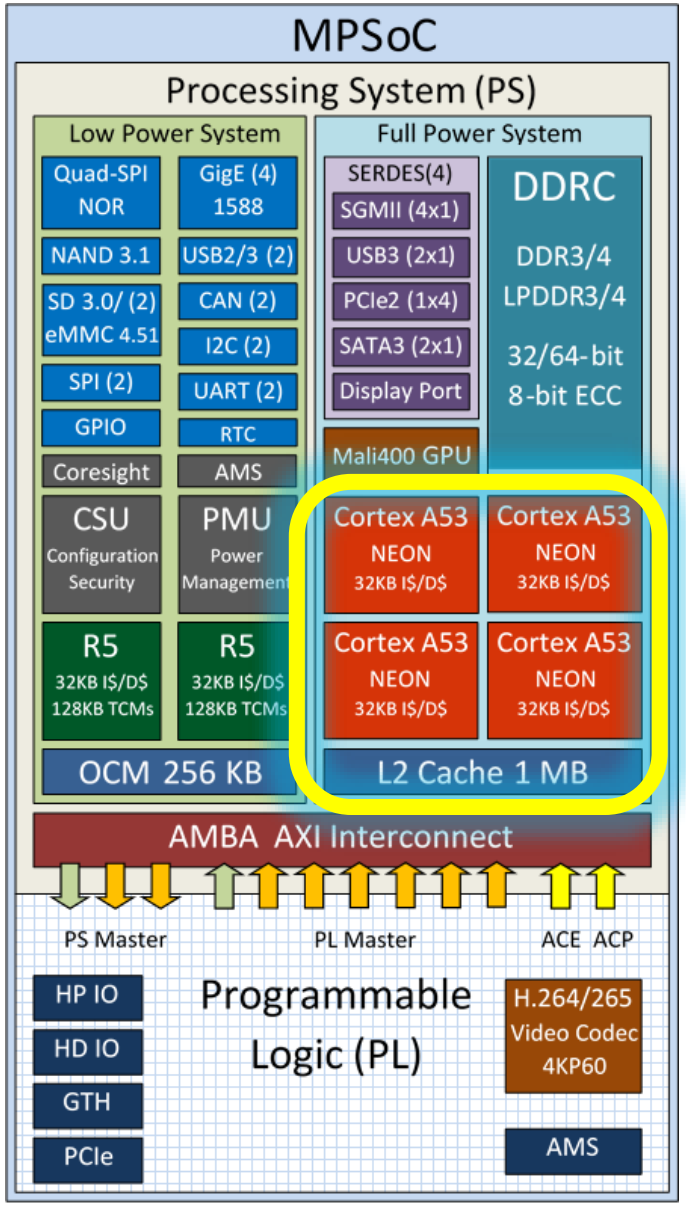
- Power efficient, 32Gbps
- 100G Ethernet and 150G Interlaken
- PCIe Gen3 & Gen4

Software & Tools

Run Time (Xilinx)	Run Time (Ecosystem)	Tools
- Linux (64b)	- FreeRTOS	- Xilinx SDK
- Hypervisor	- Micrium	- Vivado®
- OpenAMP	- WindRiver & More	- SDx environments

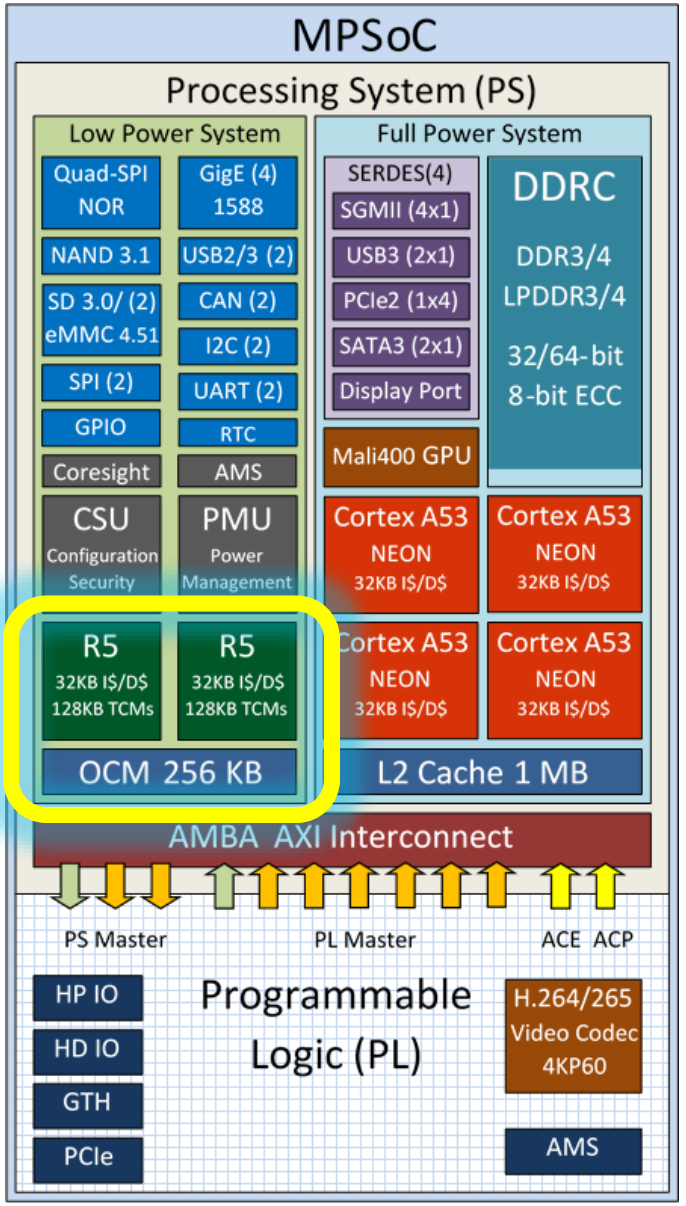
Application Processing Subsystem

- **Quad Cortex-A53 64-bit CPU**
  - 32KB each of L1 I & D$ with ECC/Parity
  - 1 MB L2 Cache with ECC
  - Virtualization Support
  - Crypto instructions support
- **IO Coherency with ACP & ACE-Lite**
- **Full Coherency between APU & PL**
- **Up to 1.5 GHz Frequency**
- **Power-gating**
  - Per core power-gating
  - L2 power-gating



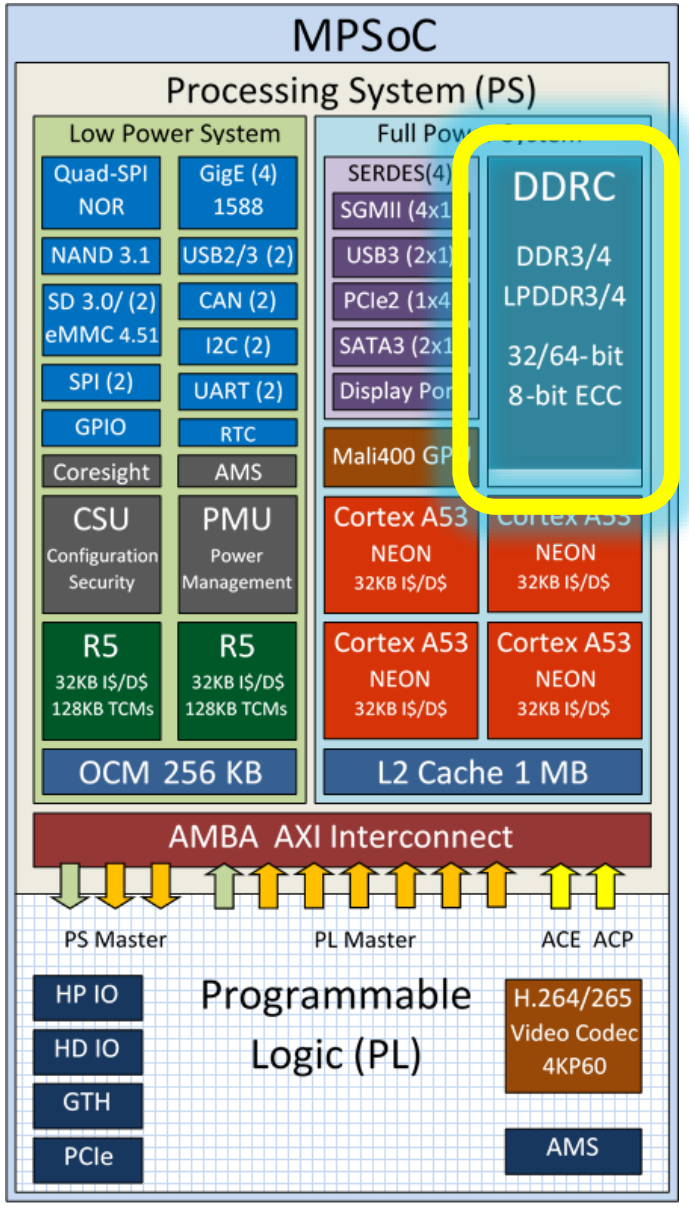
Real-time Processing Subsystem

- **Dual Core Cortex-R5 RPU**
  - Cortex-R5 Lockstep
  - Single & Double precision FPU
  - 32KB of L1 I & D caches w/ ECC
  - 256KB TCM with ECC
- **OCM (On-Chip-Memory)**
  - 256KB OCM with ECC
  - AXI Exclusive monitor support
  - Can be partitioned between different subsystems
- **Up to 600MHz Frequency**
- **Low power domain**
  - Full power domain completely powered off
  - R5s & USBs power-gate-able



Memory Subsystem

- **Six-port DDR Controller**
  - Supports exclusive monitors
- **32 or 64-bit DDR with ECC**
- **DDR3/4 and LPDDR3/4**
- **Up to 2400 Mb/s/pin**
- **QoS support for 3 traffic classes**
  - Low latency, Real-time, Best-effort
  - Guaranteed latency for RT
- **Memory protection, partitioning, and TrustZone support**
  - Using XMPU



Introducing the Zynq UltraScale+ MPSoC

ARM Cortex A53 & R5

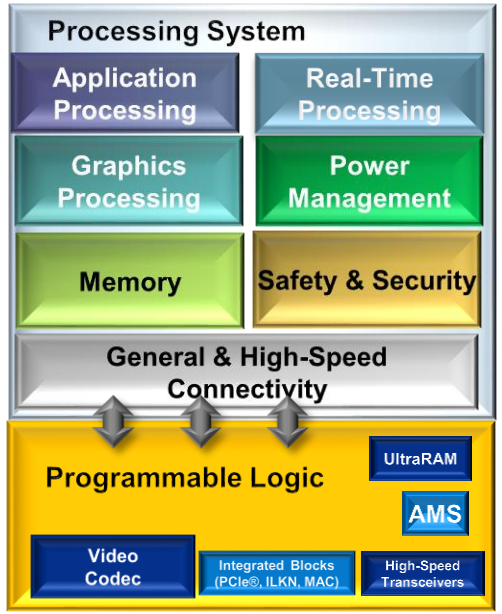
- Application processing subsystem
- Real Time processing subsystem
- Memory subsystem for max performance

IO, Video, Graphics

- Next-generation coherent interconnect
- High-speed I/Os (PCIE,USB3,SATA,GbE)
- Graphics and Video Processing Engines

Advanced Power Mgmt

- Fine-grained power reduction
- System-level software & run time opt



Security & Safety

- Enhanced Authentication, Encryption, Antitamper and trust
- Safety with industry standards support

Fabric Acceleration

- UltraScale+ fabric with time borrow
- FinFET performance and power
- HD UlltraRAM, and enhanced DSP

XCVRs & Protocols

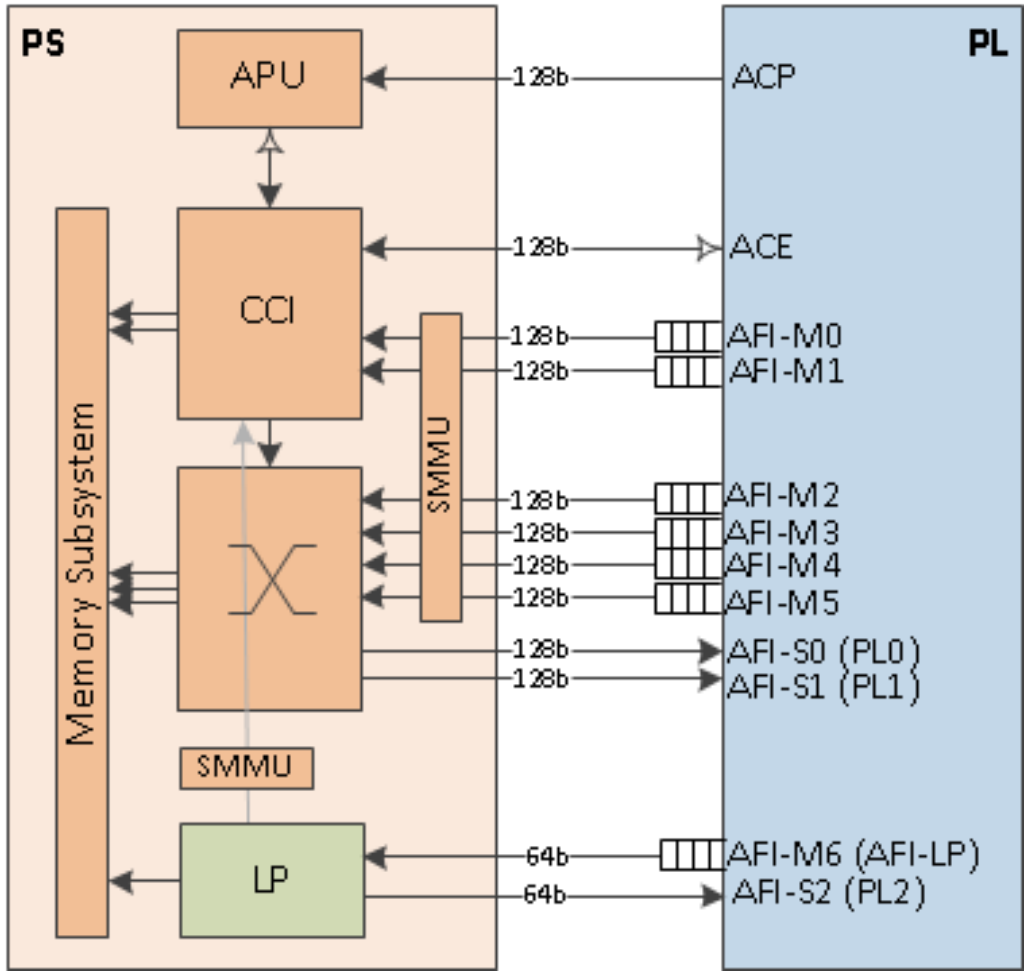
- Power efficient, 32Gbps
- 100G Ethernet and 150G Interlaken
- PCIe Gen3 & Gen4

Software & Tools

Run Time (Xilinx)	Run Time (Ecosystem)	Tools
- Linux (64b)	- FreeRTOS	- Xilinx SDK
- Hypervisor	- Micrium	- Vivado®
- OpenAMP	- WindRiver & More	- SDx environments

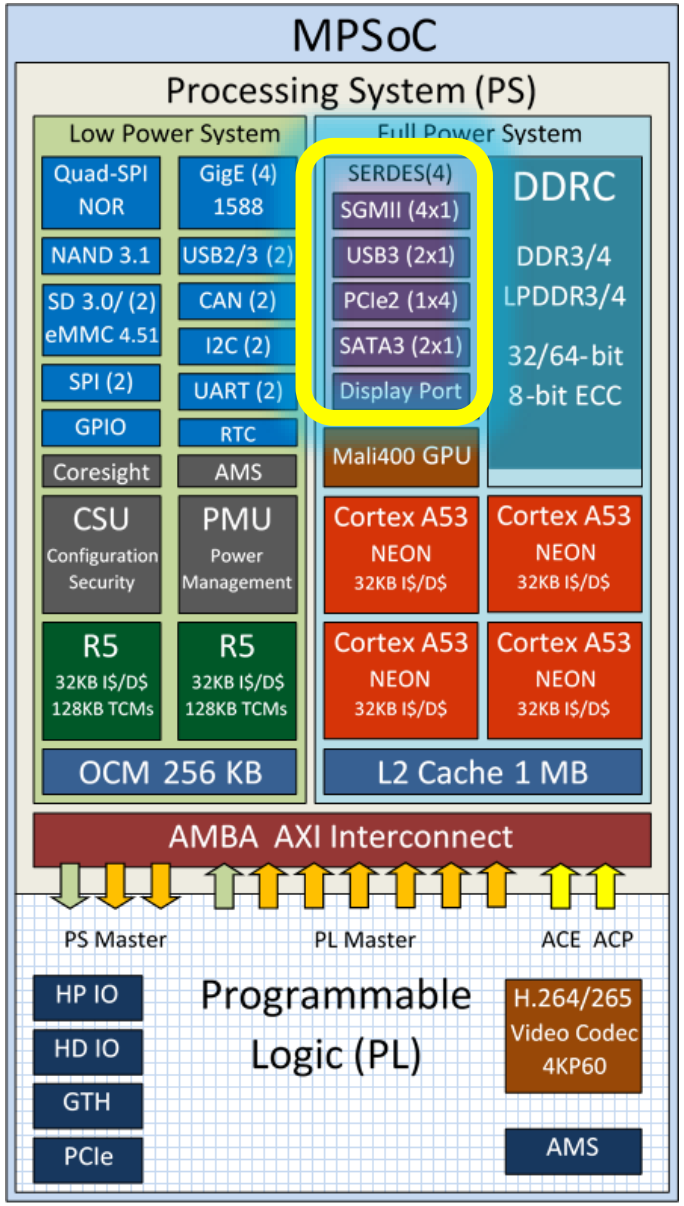
PS ⇔ PL “Data Mover” Interfaces

- **PL master ports**
  - AFI master ports
  - PL IO-coherency via CCI
  - PL virtualization via SMMU
- **PL Slave Ports**
  - PS-to-PL data movers
  - Memory mapped
- **ACP (Accelerated Coherency Port)**
  - ACP for IO (one-way) coherency
- **ACE (AXI Coherency Extensions)**
  - Full coherency between PS & PL
- **Per-port bandwidth = 85 Gb/s**
  - Read+write bandwidth



High Speed I/Os

- **USB2/3**
  - 2 independent controllers
  - OTG, Host, Device
- **SATA3**
  - Up to 2 channels
- **Display Port**
  - 4KP30 support
  - 1-2 lanes
- **PCIe Gen2 Rootport or Endpoint**
  - PCIe Gen3/4 EP also hardened
- **SGMII for GbE**
  - 4 independent GbE controllers
- **Tightly integrated transceivers**

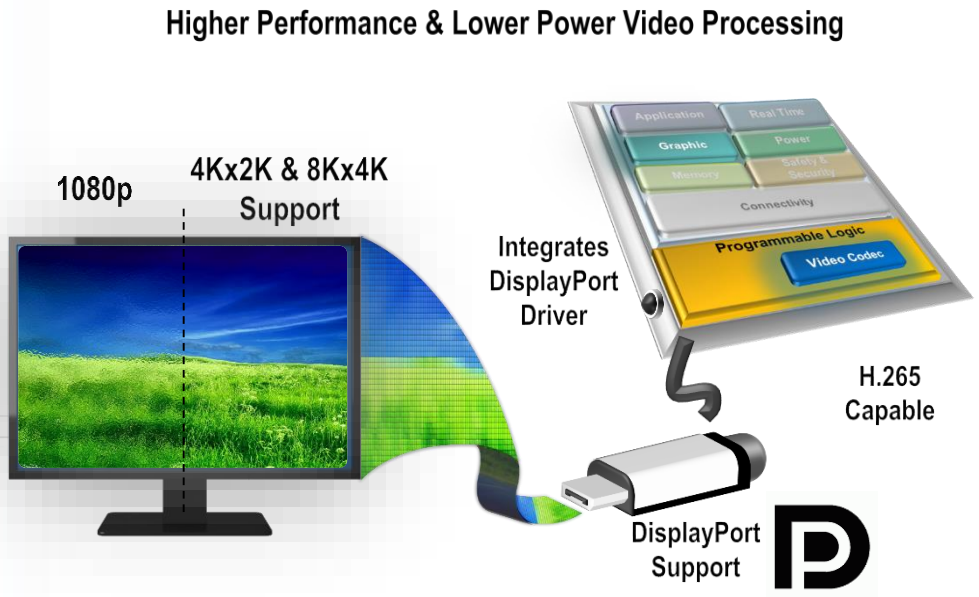


Dedicated Video Processing Engines

Graphics Processing Unit, Video CODEC & DisplayPort

Video CODEC Unit (VCU)

- More efficent vs. software implementation
  - Higher display density, faster encoding
  - Lower power consumption
- H.265 (HEVC) 8Kx4K (15 fps) 4Kx2K (60 fps)
- 8 and 10 bit per color component
- I, P, B frame support for highest compression

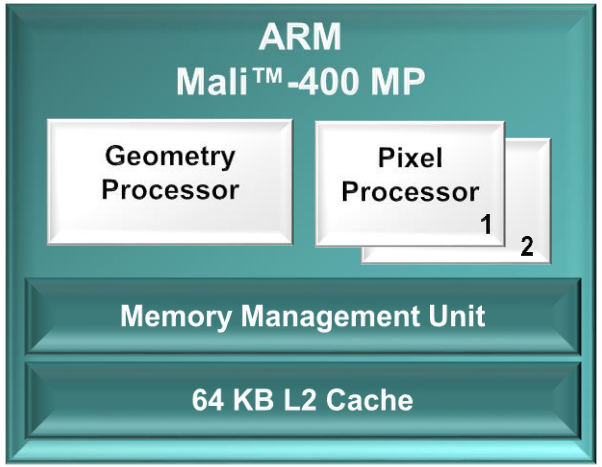


DisplayPort

- Video resolution up to 4Kx2K (30 fps)
- Audio up to 8 channels of 24-bit at up to 192 KHz
- Reducing BOM cost by eliminating display driver

Graphics Processing Unit (GPU)

- 3D visual, HMI, instrumentation, waveform display
- 1080p resolution graphics
- Mali-400 MP2 up to 667 MHz frequency (OpenGL ES 2.0, 13 GFLOPS)



Introducing the Zynq UltraScale+ MPSoC

ARM Cortex A53 & R5

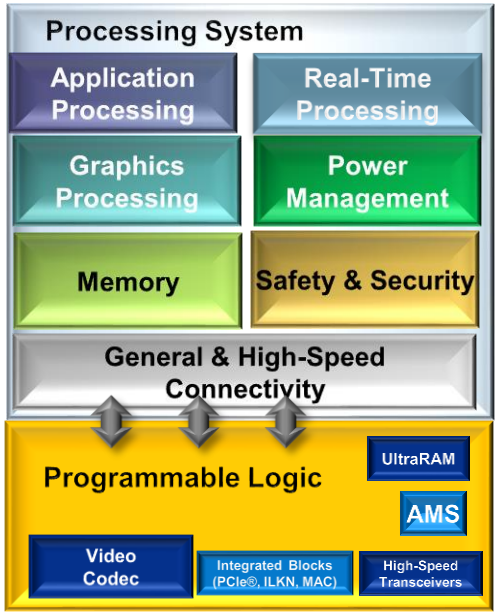
- Application processing subsystem
- Real Time processing subsystem
- Memory subsystem for max performance

IO, Video, Graphics

- Next-generation coherent interconnect
- High-speed I/Os (PCIE,USB3,SATA,GbE)
- Graphics and Video Processing Engines

Advanced Power Mgmt

- Fine-grained power reduction
- System-level software & run time opt



Security & Safety

- Enhanced Authentication, Encryption, Antitamper and trust
- Safety with industry standards support

Fabric Acceleration

- UltraScale+ fabric with time borrow
- FinFET performance and power
- HD UlltraRAM, and enhanced DSP

XCVRs & Protocols

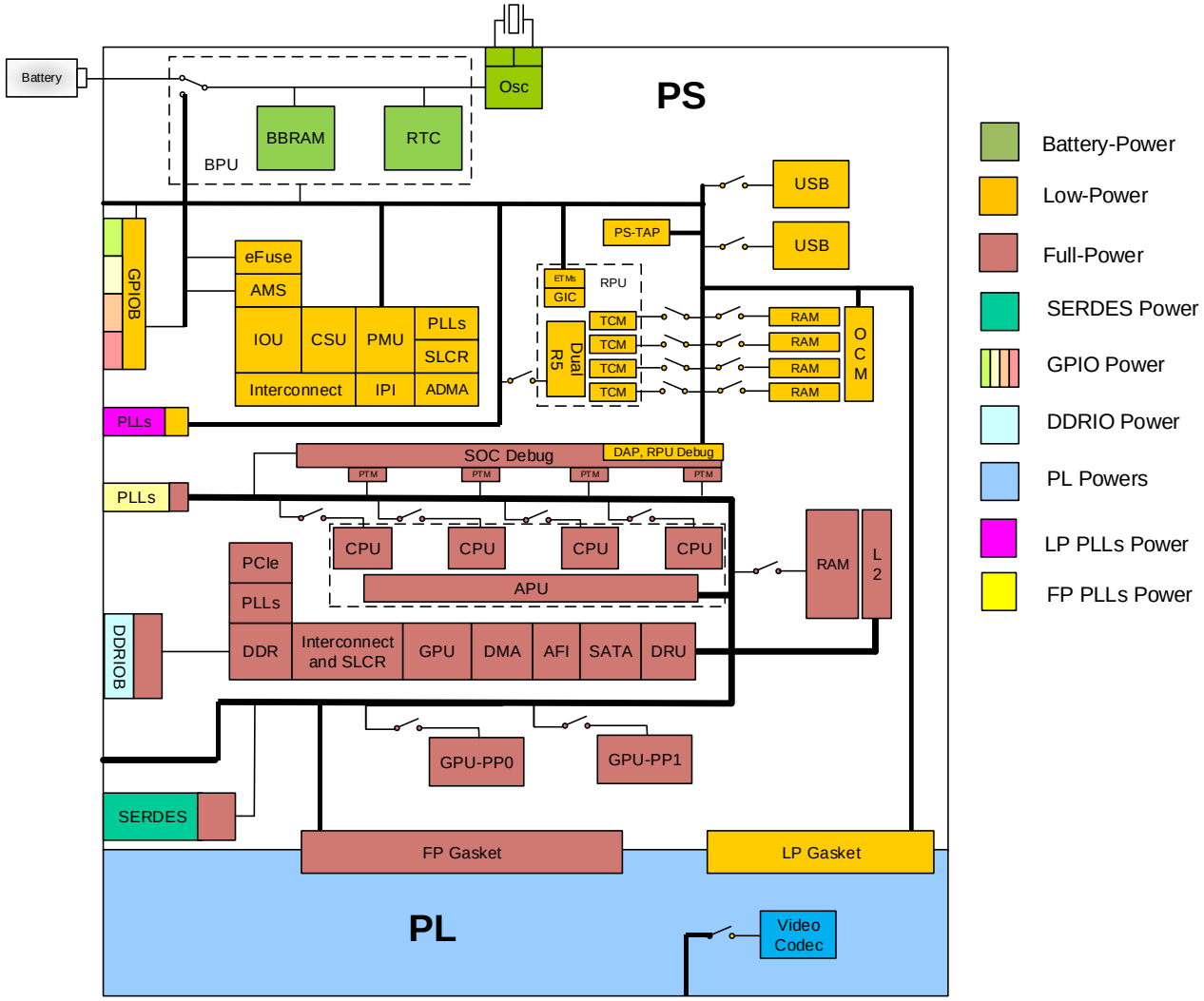
- Power efficient, 32Gbps
- 100G Ethernet and 150G Interlaken
- PCIe Gen3 & Gen4

Software & Tools

Run Time (Xilinx)	Run Time (Ecosystem)	Tools
- Linux (64b)	- FreeRTOS	- Xilinx SDK
- Hypervisor	- Micrium	- Vivado®
- OpenAMP	- WindRiver & More	- SDx environments

Power-Domains and Power-Gating

- **Multiple power domains**
  - Low power domain
  - Full power domain
  - PL power domain
- **Power gating**
  - A53 per core
  - L2 and OCM RAM
  - GPU, USB
  - R5s & TCM
  - Video CODEC
- **Sleep Mode**
  - 35mW sleep mode
  - Suspend to DDR with power off



Security, Safety & Reliability

Advanced Device-Level Secure Processing

- Information Assurance, Anti-Tamper, Trust
- Multi-layered Authentication for Secure System Boot
- Key Management & Revocation

Architected for Safe Systems

- IEC61508 & ISO26262 Functional Safety Standards
- Redundancy, Diversity and Lock-step
- Layered Partitioning: Core / Infrastructure / Peripherals

Delivering High Reliability

- High Availability Systems
- Error Detection & Handling
- Subsystem Isolation & Protection

Introducing the Zynq UltraScale+ MPSoC

ARM Cortex A53 & R5

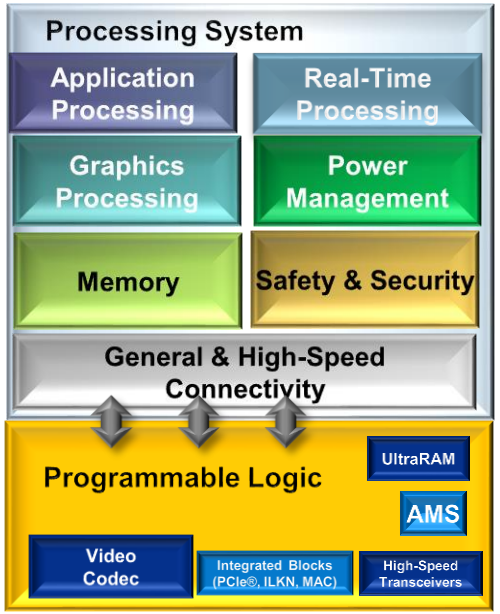
- Application processing subsystem
- Real Time processing subsystem
- Memory subsystem for max performance

IO, Video, Graphics

- Next-generation coherent interconnect
- High-speed I/Os (PCIE,USB3,SATA,GbE)
- Graphics and Video Processing Engines

Advanced Power Mgmt

- Fine-grained power reduction
- System-level software & run time opt



Security & Safety

- Enhanced Authentication, Encryption, Antitamper and trust
- Safety with industry standards support

Fabric Acceleration

- UltraScale+ fabric with time borrow
- FinFET performance and power
- HD UlltraRAM, and enhanced DSP

XCVRs & Protocols

- Power efficient, 32Gbps
- 100G Ethernet and 150G Interlaken
- PCIe Gen3 & Gen4

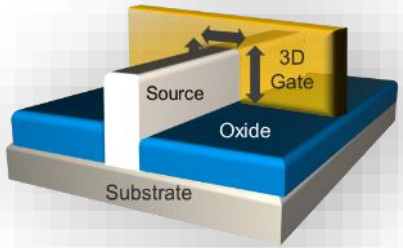
Software & Tools

Run Time (Xilinx)	Run Time (Ecosystem)	Tools
- Linux (64b)	- FreeRTOS	- Xilinx SDK
- Hypervisor	- Micrium	- Vivado®
- OpenAMP	- WindRiver & More	- SDx environments

Tuned Process for Optimal Performance/Watt

Optimal Operating Voltage Selection

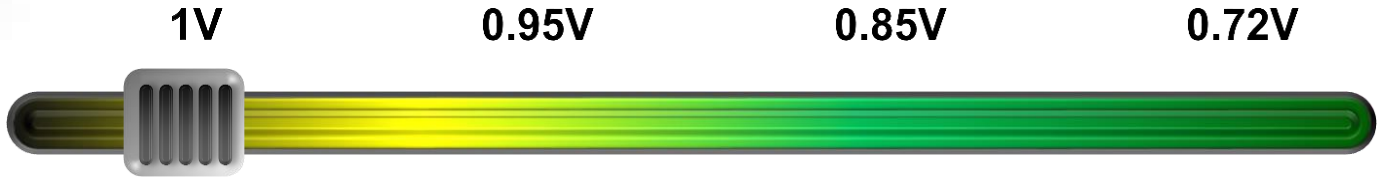
3D FinFET



3D Gate "wraps" around channel for more surface area, achieving

- ✓ Faster transistor on/off switching speeds for greater performance
- ✓ Lower leakage and operating voltage for lower power

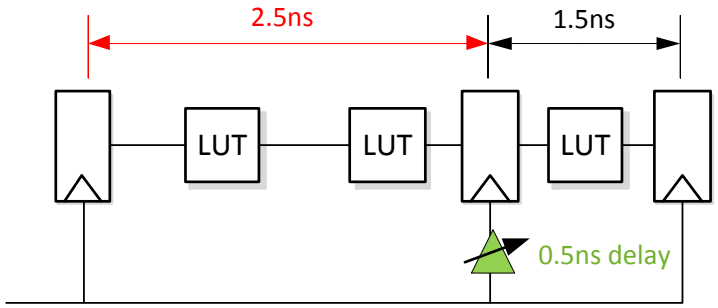
Tuned Operating Voltage (V_{CCINT})



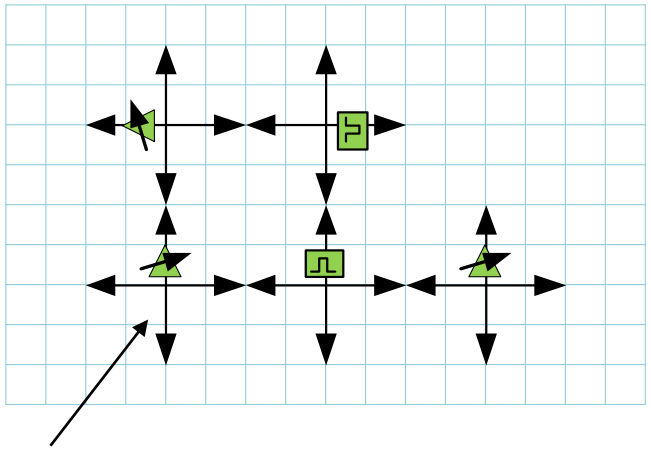
	7 Series (28nm) V_{NOM}	20% Greater Performance	UltraScale (20nm) V_{NOM}	33% Greater Performance	UltraScale+ (16nm) V_{NOM}	UltraScale+ (16nm) V_{LOW}
Normalized Fabric Performance	1.0x		1.2x		1.6x	1.2x
Normalized Total Power	1.0x		.7x		.8x	.5x
Performance/Watt	1.0x		1.7x		2x	2.4x

Time Borrow in the Fabric

- **Time-borrowing concept**
  - Shift available slack from fast stages to performance-critical paths

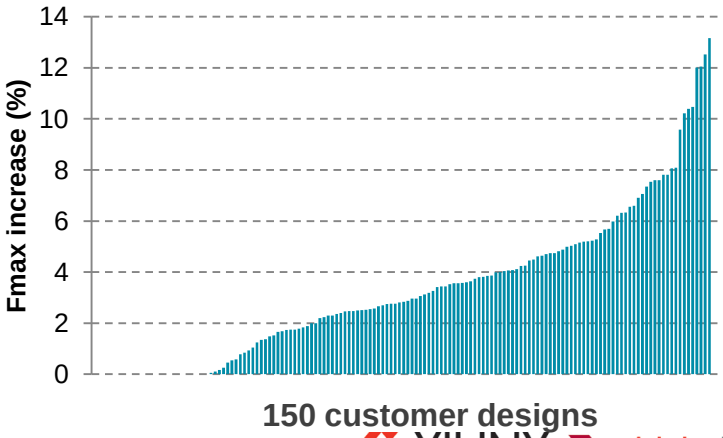


Example	Tmin	Fmax
Baseline	2.5 ns	400 MHz
Time Borrow	2 ns	500 MHz



- **UltraScale+ time-borrowing platform**
  - Fine-grain delays to adjust clock skew
  - Programmable pulse generators for latch-based time borrow

- **High Performance without design changes**
  - Very effective on high-performance designs
  - Transparent to customers, part of default flow

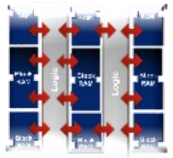


UltraRAM: New Memory Technology

Distributed RAM
(bits to kilobits)

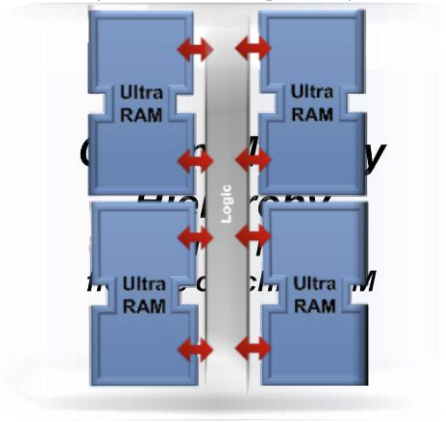
- Wide, shallow FIFOs
- Shift registers
- State machines

Block RAM
(10s of megabits)



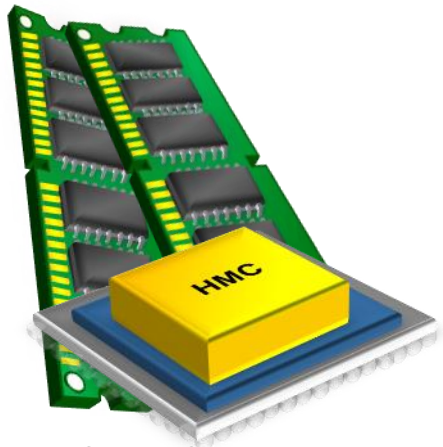
- Data/coefficient storage
- Deep FIFOs
- Shallow buffering

UltraRAM
(100s of megabits)



- Deep packet buffering
- Video buffering
- State, statistics, counters

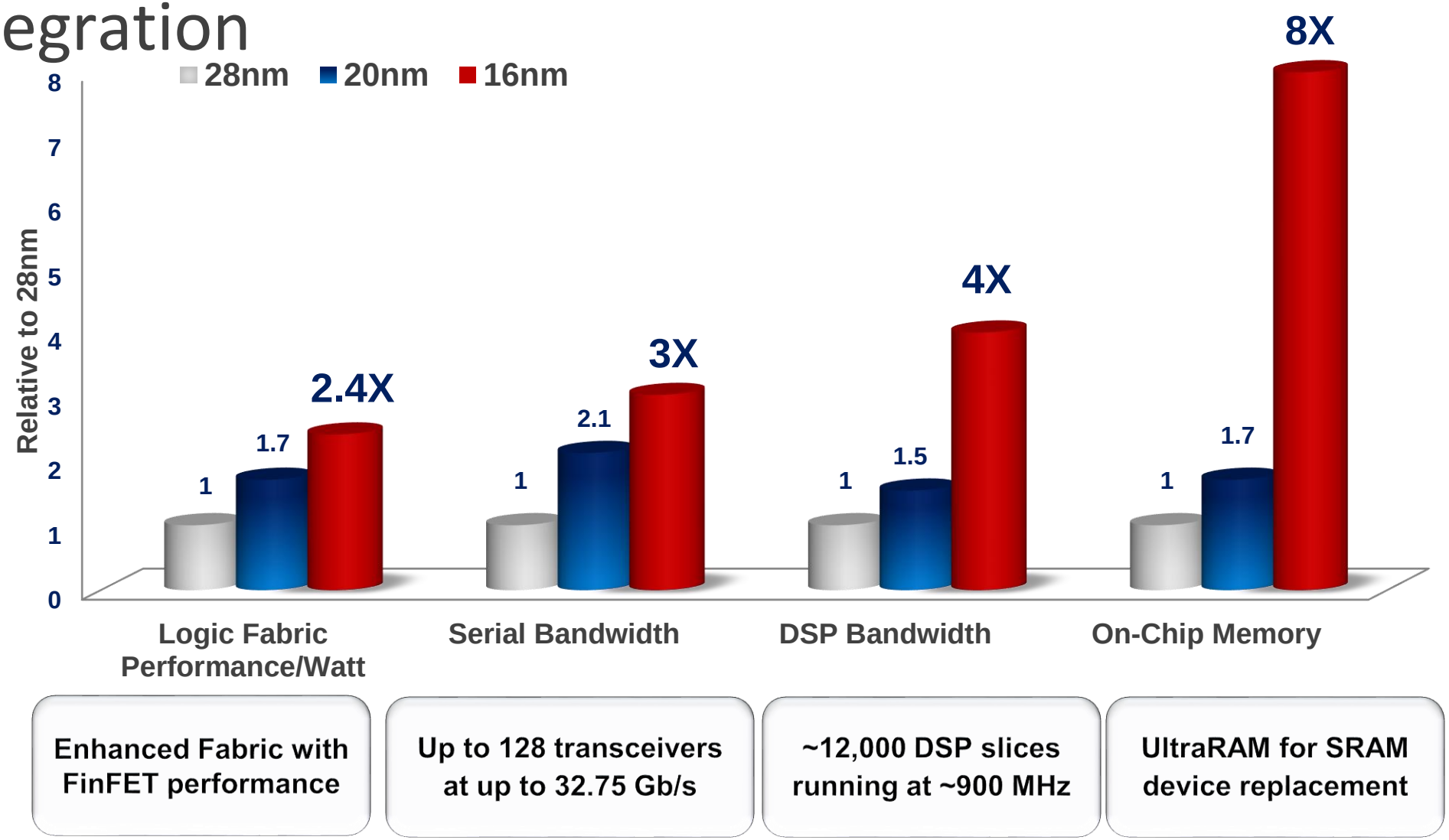
External Memory
(100s of megabits to gigabits)



Larger data storage

Up to 432 Mb to replace external memory for cost, power, performance

Unlocking Performance, Bandwidth, & Integration



Introducing the Zynq UltraScale+ MPSoC

ARM Cortex A53 & R5

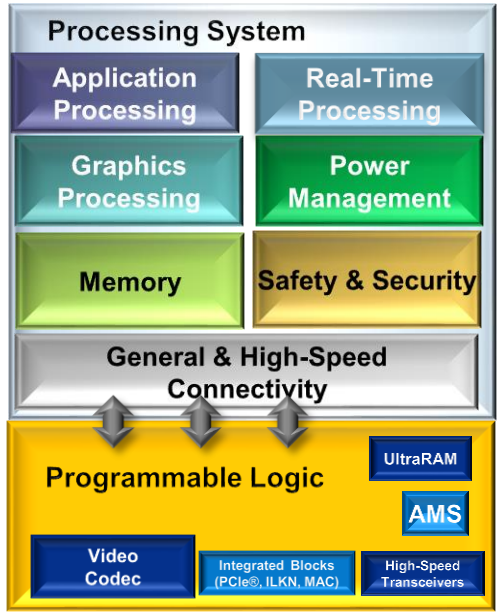
- Application processing subsystem
- Real Time processing subsystem
- Memory subsystem for max performance

IO, Video, Graphics

- Next-generation coherent interconnect
- High-speed I/Os (PCIE,USB3,SATA,GbE)
- Graphics and Video Processing Engines

Advanced Power Mgmt

- Fine-grained power reduction
- System-level software & run time opt



Security & Safety

- Enhanced Authentication, Encryption, Antitamper and trust
- Safety with industry standards support

Fabric Acceleration

- UltraScale+ fabric with time borrow
- FinFET performance and power
- HD UlltraRAM, and enhanced DSP

XCVRs & Protocols

- Power efficient, 32Gbps
- 100G Ethernet and 150G Interlaken
- PCIe Gen3 & Gen4

Software & Tools

Run Time (Xilinx)	Run Time (Ecosystem)	Tools
- Linux (64b)	- FreeRTOS	- Xilinx SDK
- Hypervisor	- Micrium	- Vivado®
- OpenAMP	- WindRiver & More	- SDx environments

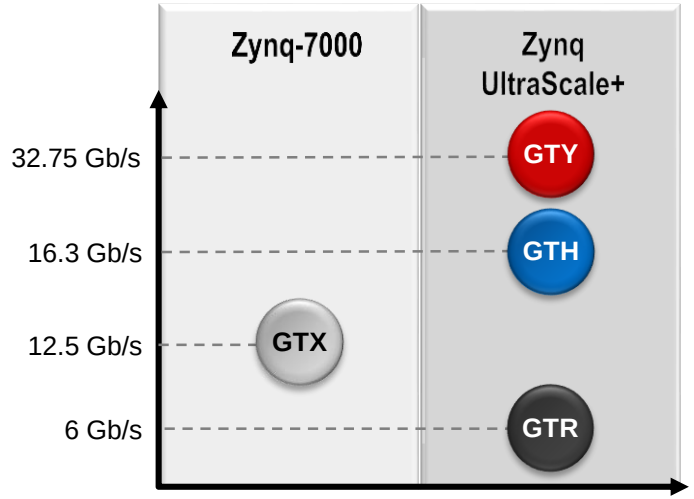
Enhanced transceivers

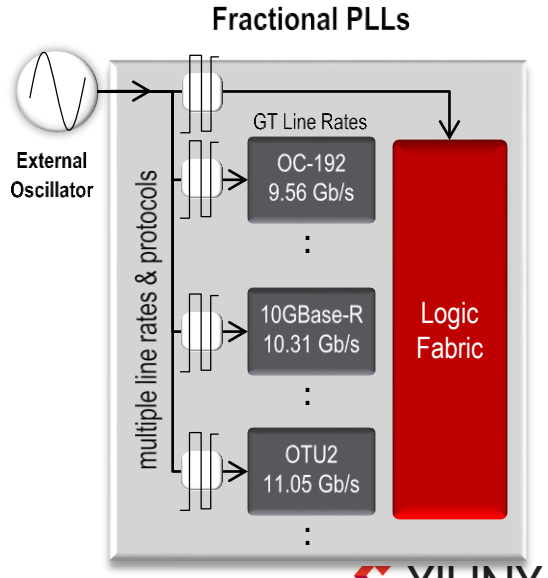
Diverse, Power Efficient SerDes for Bandwidth

- 16G (GTH) & 32G (GTY) transceivers in PL,
- 6G (GTR) in PS for direct access to key processing elements, with full PHY/IP compliance for key protocols:
  - USB, SATA, DisplayPort, PCIe, Ethernet

Fractional PLLs to Reduce BOM Cost

- Single external oscillator generates GT & logic fabric clocks for multiple non-integer line rates
- Available in GTH, and GTY transceivers





Introducing the Zynq UltraScale+ MPSoC

ARM Cortex A53 & R5

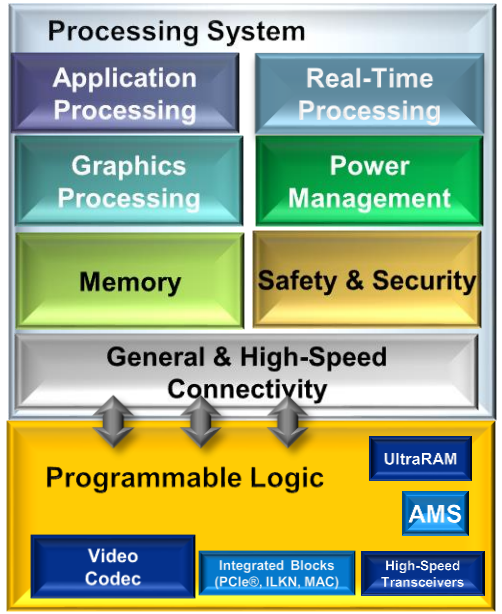
- Application processing subsystem
- Real Time processing subsystem
- Memory subsystem for max performance

IO, Video, Graphics

- Next-generation coherent interconnect
- High-speed I/Os (PCIE,USB3,SATA,GbE)
- Graphics and Video Processing Engines

Advanced Power Mgmt

- Fine-grained power reduction
- System-level software & run time opt



Security & Safety

- Enhanced Authentication, Encryption, Antitamper and trust
- Safety with industry standards support

Fabric Acceleration

- UltraScale+ fabric with time borrow
- FinFET performance and power
- HD UlltraRAM, and enhanced DSP

XCVRs & Protocols

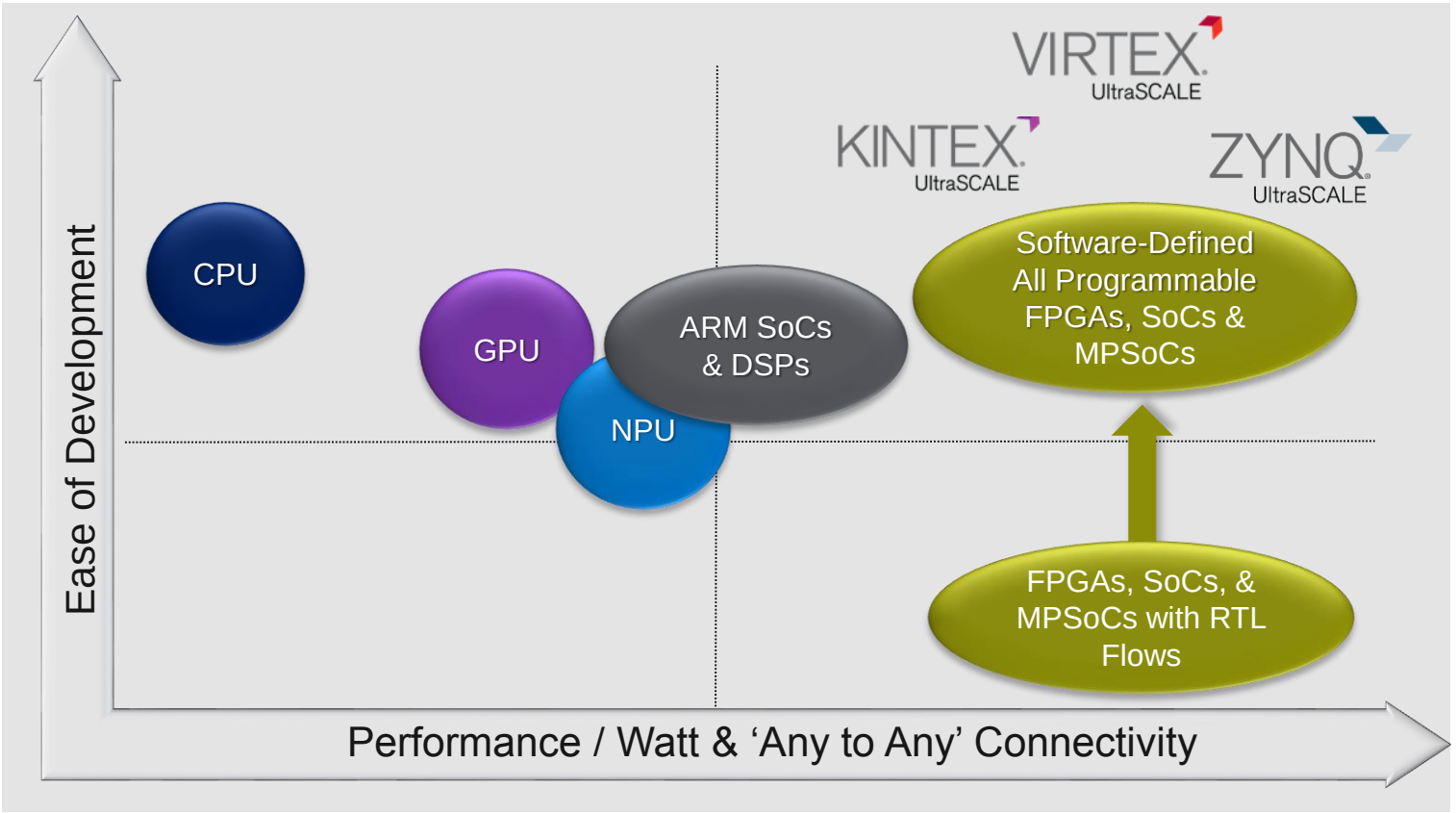
- Power efficient, 32Gbps
- 100G Ethernet and 150G Interlaken
- PCIe Gen3 & Gen4

Software & Tools

Run Time (Xilinx)	Run Time (Ecosystem)	Tools
- Linux (64b)	- FreeRTOS	- Xilinx SDK
- Hypervisor	- Micrium	- Vivado®
- OpenAMP	- WindRiver & More	- SDx environments

High-Level Programming Tools

Smarter Systems Compute Acceleration Options

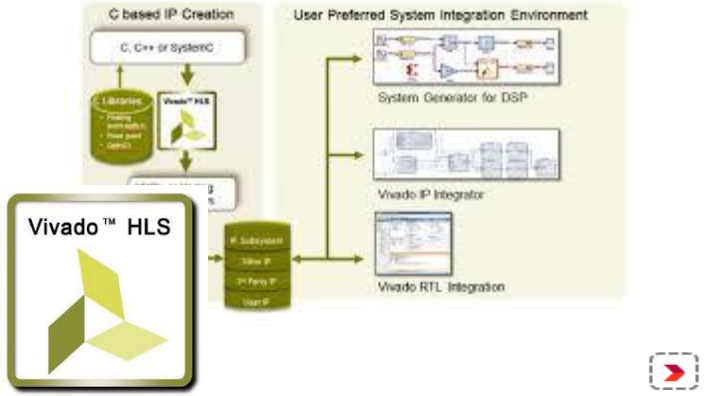


Note: Software programmable devices often paired with FPGA for connectivity and co-processing

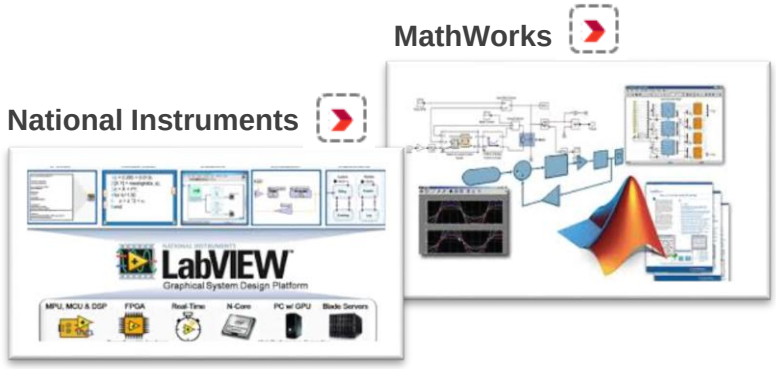
Enabling Smarter Systems

Abstractions

Software Automation



Hardware Automation



System Automation

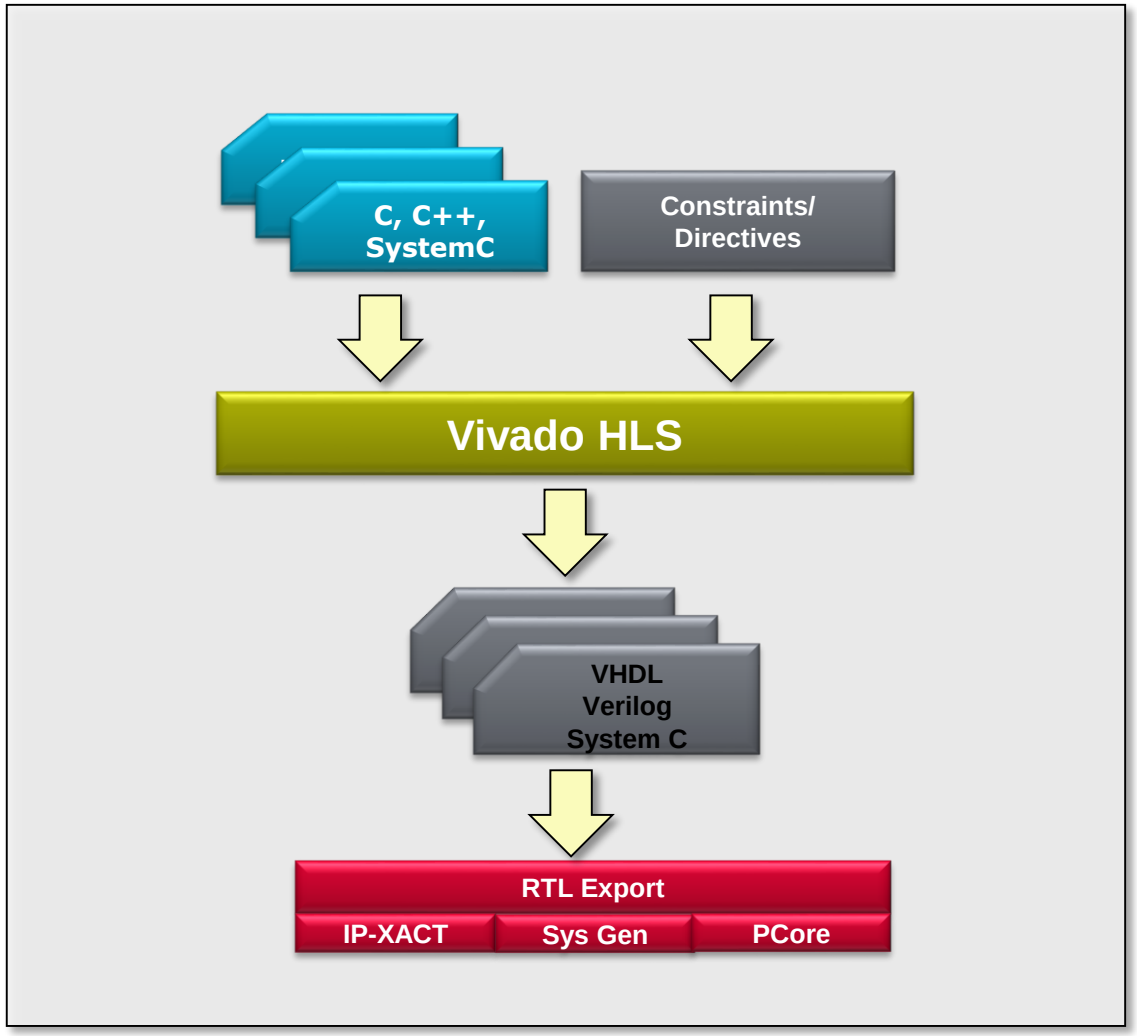
Bottom-up Agenda

- Vivado HLS: the core
- System integration level
  - SDSoC: replace SoC
  - SDAccel: replace CPU/GPU
  - SDNet: replace NPU/ASSP
- Application example: Memcached

Vivado HLS: High-Level Synthesis

High-Level Synthesis: HLS

- **High-Level Synthesis**
  - Creates an RTL implementation from C level source code
  - Extracts control and dataflow from the source code
  - Implements the design based on defaults and user applied directives
- **Many implementation are possible from the same source description**
  - Smaller designs, faster designs, optimal designs
  - Enables design exploration



Design Exploration with Directives

One body of code:
Many hardware outcomes

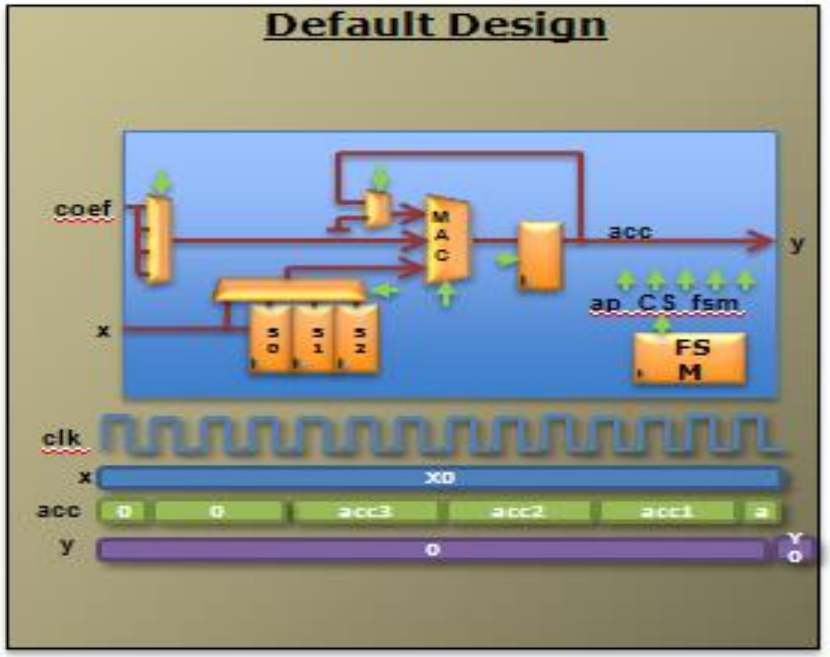
```
…
loop: for (i=3;i>=0;i--) {
  if (i==0) {
    acc+=x*c[0];
    shift_reg[0]=x;
  } else {
     shift_reg[i]=shift_reg[i-1];
     acc+=shift_reg[i]*c[i];
  }
}
….
```

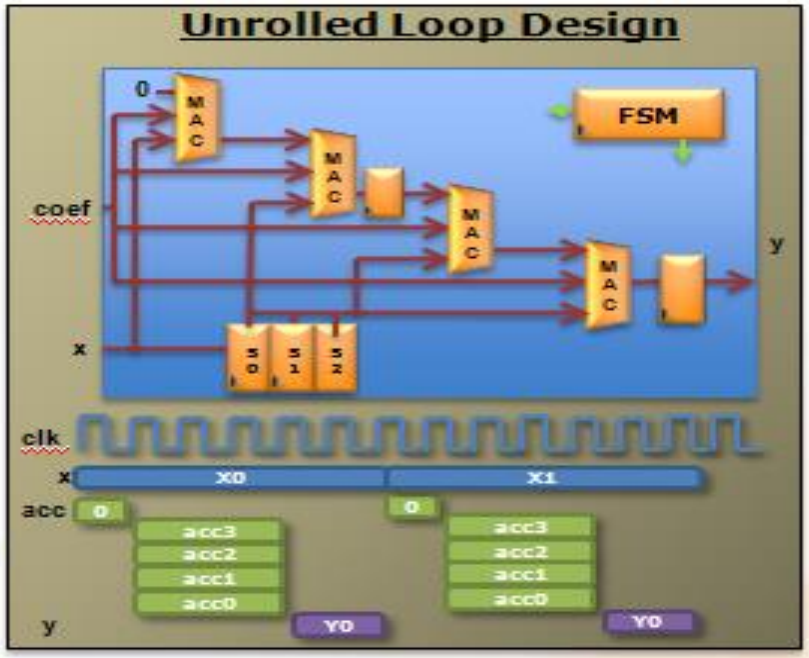
Before we get into details, let's look under the hood ….

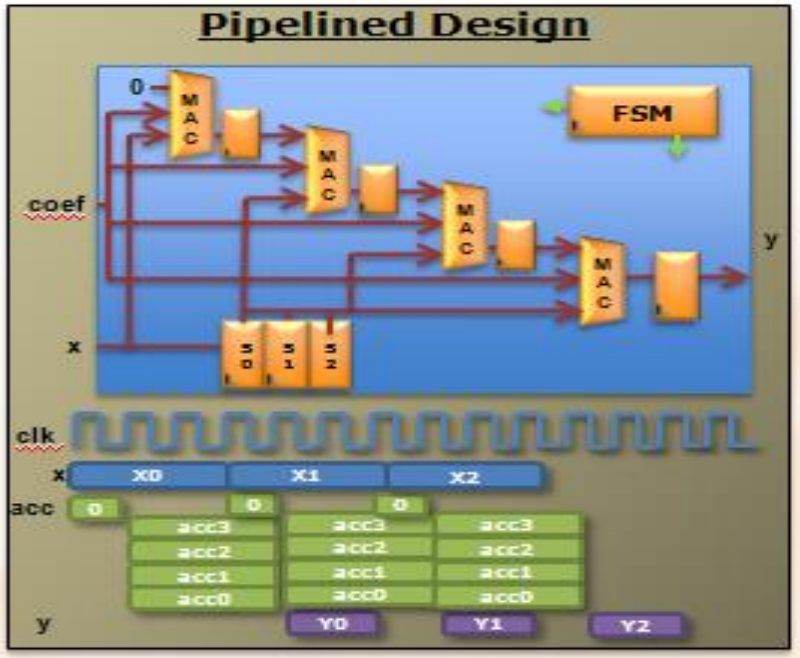
The same hardware is used for each iteration of the loop:
- **Small area**
- **Long latency**
- **Low throughput**

Different hardware is used for each iteration of the loop:
- **Higher area**
- **Short latency**
- **Better throughput**

Different iterations are executed concurrently:
- **Higher area**
- **Short latency**
- **Best throughput**





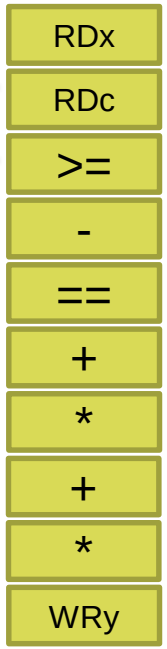


HLS: Control & Datapath Extraction

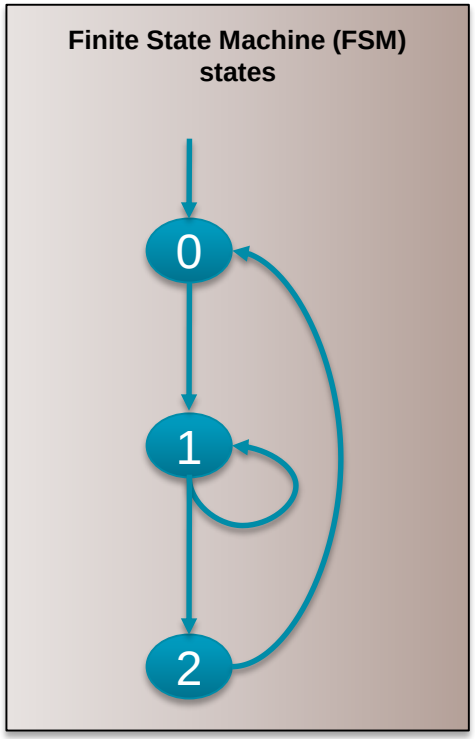
Code

```
void fir (
  data_t *y,
  coef_t c[4],
  data_t x
  ) {

  static data_t shift_reg[4];
  acc_t acc;
  int i;

  acc=0;
  loop: for (i=3;i>=0;i--) {
    if (i==0) {
      acc+=x*c[0];
      shift_reg[0]=x;
    } else {
       shift_reg[i]=shift_reg[i-1];
       acc+=shift_reg[i]*c[i];
    }
  }
  *y=acc;
}
```

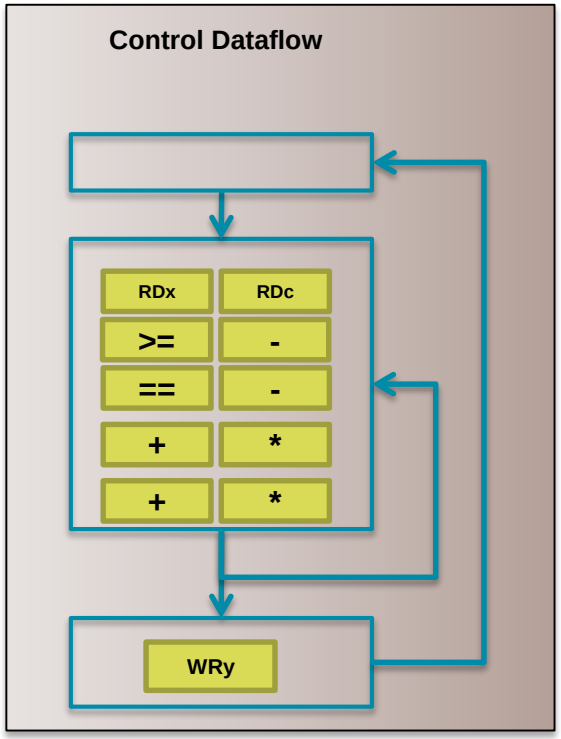
Operations



Control Behavior



Control & Datapath Behavior



From any C code example ..

Operations are extracted…

The control is known

A unified control dataflow behavior is created.

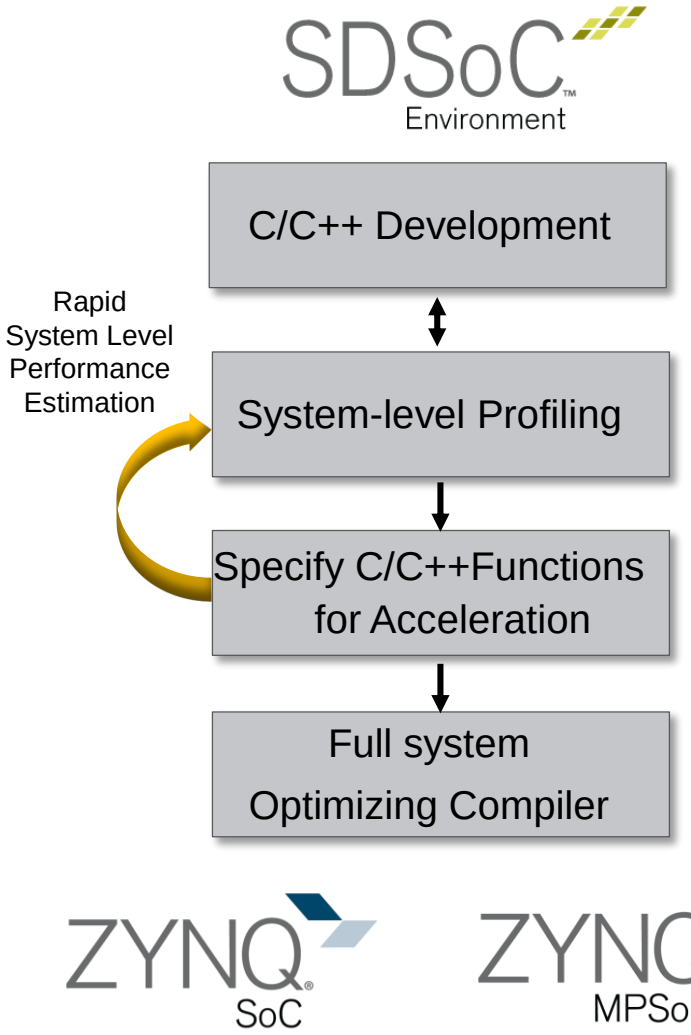
Summary

- **In HLS**
  - C becomes RTL
  - Operations in the code map to hardware resources
  - Understand how constructs such as functions, loops and arrays are synthesized
- **HLS design involves**
  - Synthesize the initial design
  - Analyze to see what limits the performance
    - User directives to change the default behaviors
    - Remove bottlenecks
  - Analyze to see what limits the area
    - The types used define the size of operators
    - This can have an impact on what operations can fit in a clock cycle
- **Use directives to shape the initial design to meet performance**
  - Increase parallelism to improve performance
  - Refine bit sizes and sharing to reduce area

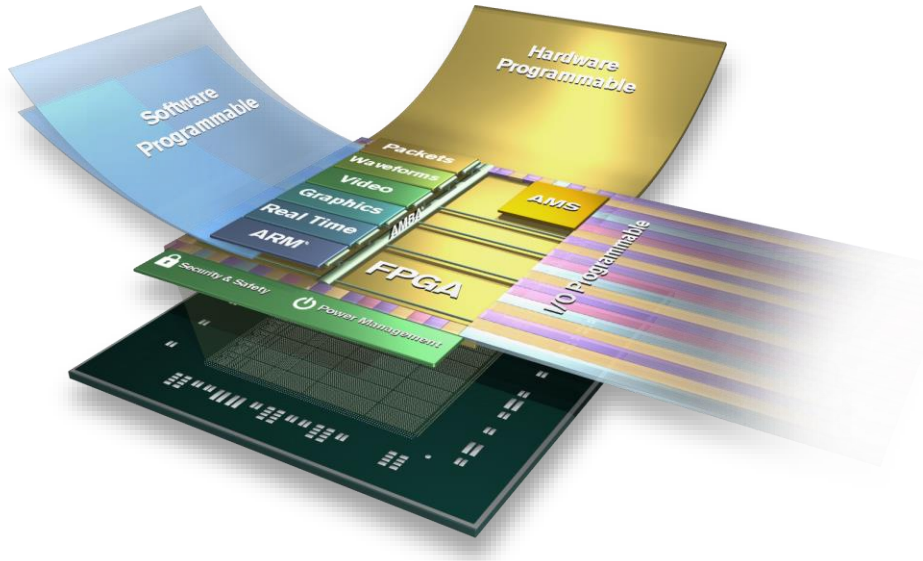
SDSoC

SDSoC Development Environment

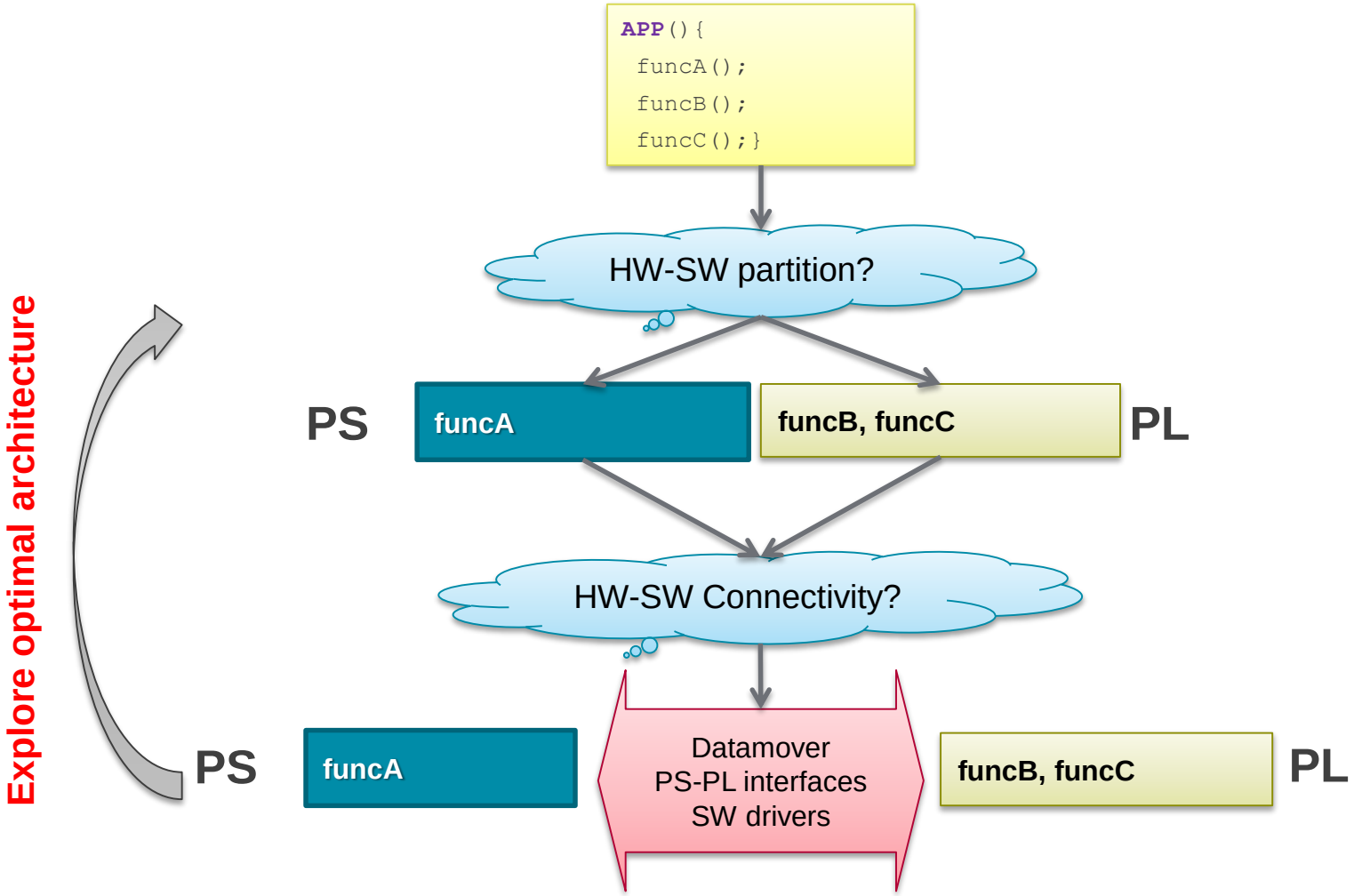
C/C++ Programming for Zynq SoC and MPSoC



- ASSP-like programming experience
- System-level profiling
- Full system optimizing compiler
- Expert use model for platform developers and system architects

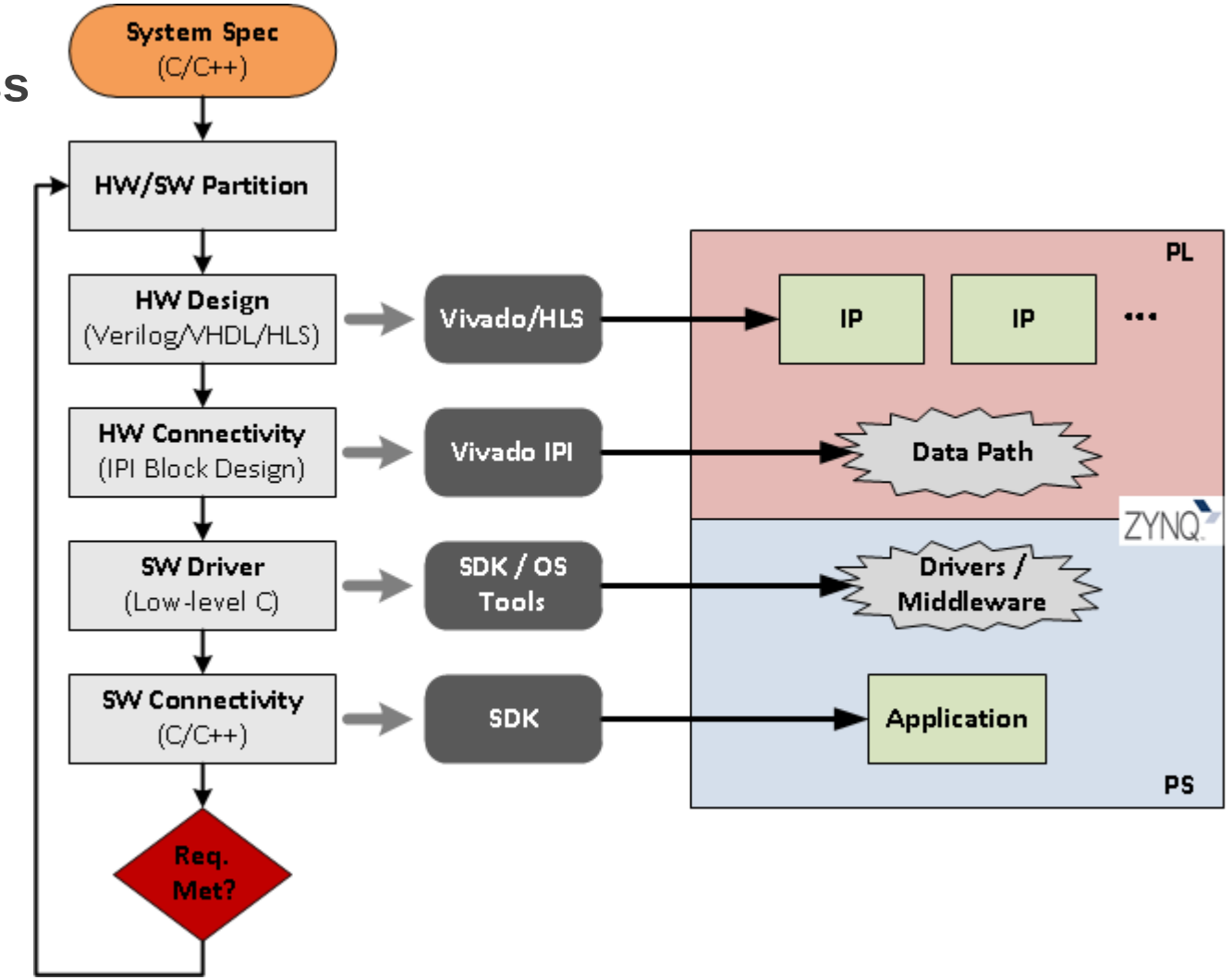


All Programmable SoCs Development Flow



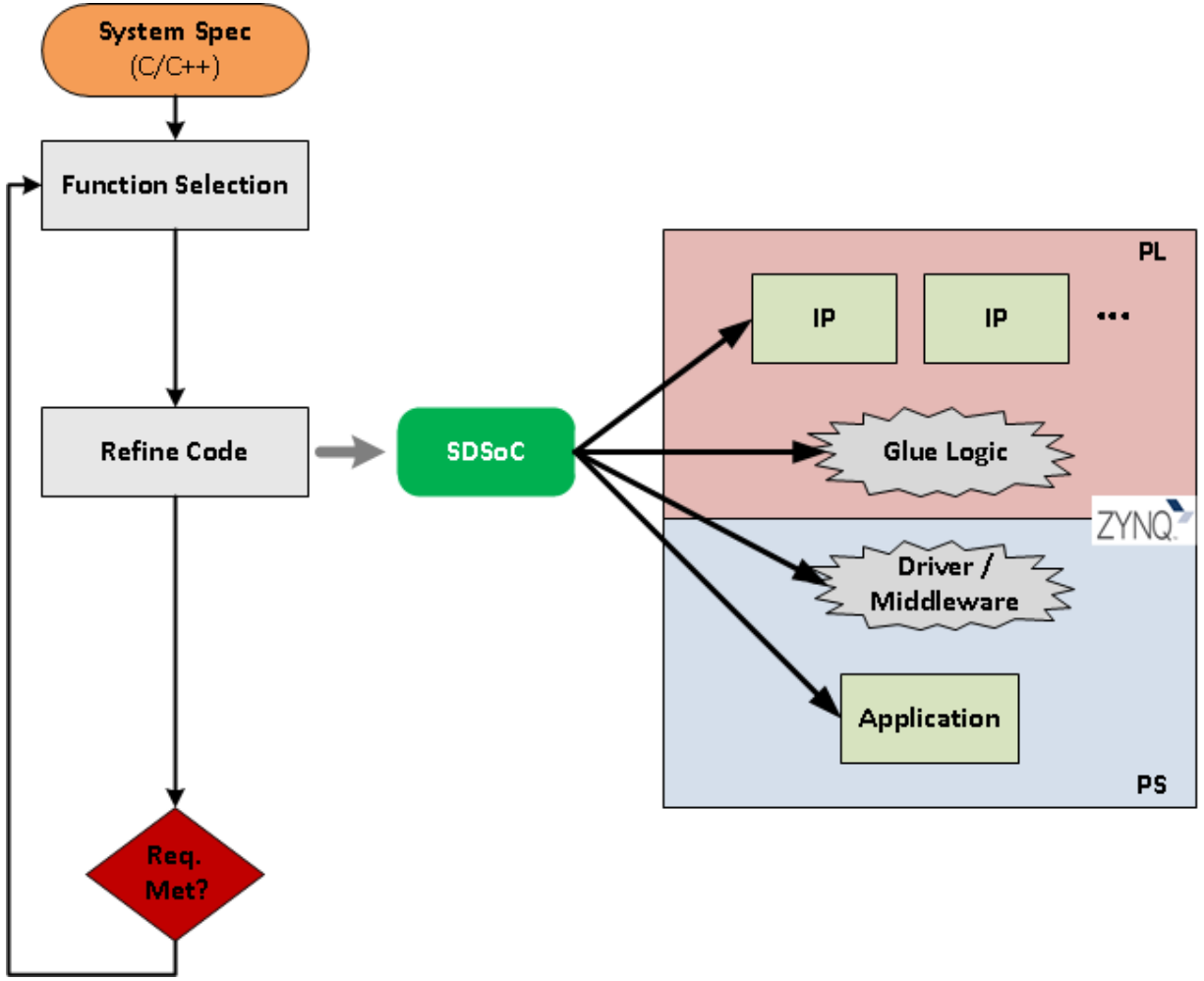
Development Flow Without SDSoC

- **Overall process requires expertise at all steps in the development process**
  - Various hardware design entries
  - Hardware connectivity at system level
  - Driver development to drive custom hardware
  - Integrating with application and target OS



SDSoC Development Environment

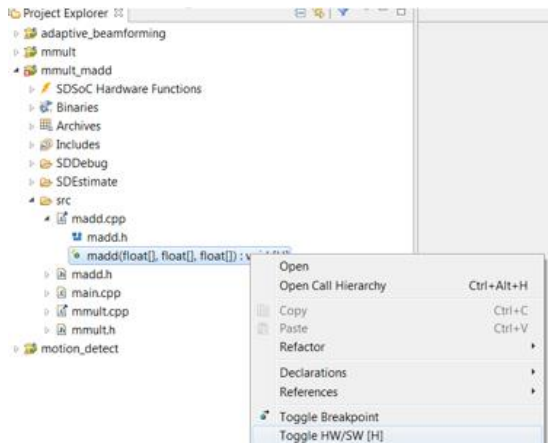
- **SDSoC development environment consolidates a multi-step/multi-tool process into a single tool and reduces hardware/software partitioning to simple function selection**
  - Code typically needs to be refined to achieve optimal results



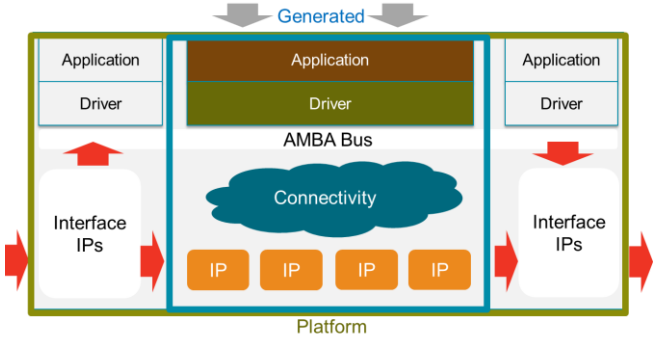
Software Development Environment

C/C++ Application

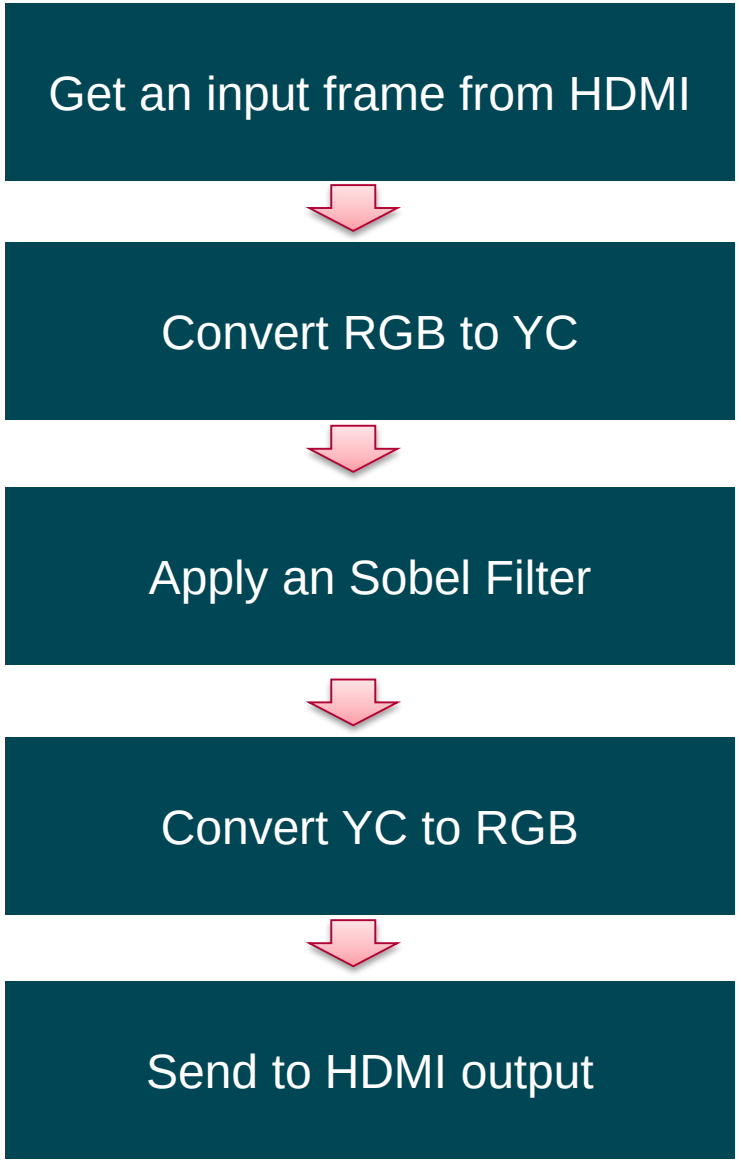
Estimator | **Compiler** | **Debugger** | **Profiler**



Leverages proven Xilinx tools in the backend



Optimization of Sobel filtering on video stream



Reference application

```
void img_process(unsigned int *rgb_data_in,
                 unsigned int *rgb_data_out,
                 unsigned short *yc_data_in,
                 unsigned short *yc_sobel_out)
{

    rgb_pad2ycbcr(rgb_data_in, yc_data_in);
    sobel_filter(yc_data_in, yc_sobel_out);
    ycbcr2rgb_pad(yc_sobel_out, rgb_data_out);
}
```

Naïve Sobel Filter Implementation

```
void sobel_filter(unsigned short *yc_in,unsigned short *yc_out, short *x_op, short *y_op)
{
  int row, col;
  for(row = 0; row < NUMROWS; row++){
    for(col = 0; col < NUMCOLS; col++){
      unsigned short input_data, unsigned char edge;
      if((col < NUMCOLS) & (row < NUMROWS))
        input_data = yc_in[row*NUMCOLS+col];
      if( isEdge(row, col)) edge=0;
      else{
        short x_weight = 0, y_weight = 0;
        for(char I = 0; i < 3; i++){
          for(char j = 0; j < 3; j++){
            unsigned short temp = (yc_in[index(row, col, I, j)]);
            x_weight += (temp * x_op[i][j]);
            y_weight += (temp * y_op[i][j]);
        } }
        edge = ABS(x_weight) + ABS(y_weight);
      }
      yc_out[index(row, col)] = edge;
    }
  }
}
```

Iterate over an input video image

Applying 3x3 sobel filter

Writing to output image

Video 1: Run the Naïve Sobel on a Zynq board

- **0.1 FPS @ 1080p**

Problem 1: Non-Sequential Overlapped Memory Access

```
void sobel_filter(unsigned short *yc_in,unsigned short *yc_out, short *x_op, short *y_op)
{
  int row, col;
  for(row = 0; row < NUMROWS; row++){
    for(col = 0; col < NUMCOLS; col++){
      unsigned short input_data, unsigned char edge;
      if((col < NUMCOLS) & (row < NUMROWS))
        input_data = yc_in[row*NUMCOLS+col];
      if( isEdge(row, col)) edge=0;
      else{
        short x_weight = 0, y_weight = 0;
        for(char I = 0; i < 3; i++){
          for(char j = 0; j < 3; j++){
            unsigned short temp = (yc_in[index(row, col, I, j)]);
            x_weight += (temp * x_op[i][j]);
            y_weight += (temp * y_op[i][j]);
        } }
        edge = ABS(x_weight) + ABS(y_weight);
      }
      yc_out[index(row, col)] = edge;
    }
  }
}
```

9 memory access per iteration

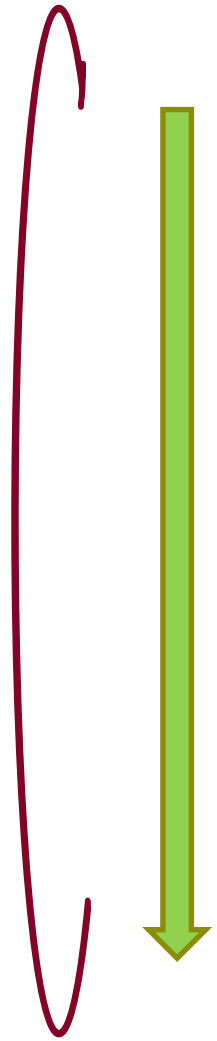
Solution 1: Use ap_linebuffer and ap_window Classes

```
void sobel_filter(unsigned short *yc_in,unsigned short *yc_out, short *x_op, short *y_op)
{
  int row, col;
  ap_linebuffer<unsigned char, 3, NUMCOLS> buff_A;
  ap_window<unsigned char,3,3> buff_C;
  for(row = 0; row < NUMROWS; row++){
    for(col = 0; col < NUMCOLS; col++){
      unsigned short input_data, unsigned char edge;
      update_linebuffer(buff_A, yc_in[index(row,col)]);
      update_window(buff_C);
      if((col < NUMCOLS) & (row < NUMROWS))
        input_data = yc_in[row*NUMCOLS+col];
      if( isEdge(row, col)) edge=0;
      else{
        short x_weight = 0, y_weight = 0;
        for(char I = 0; i < 3; i++){
          for(char j = 0; j < 3; j++){
            unsigned short temp = (buff_C[index(row, col, I, j)]);
            x_weight += (temp * x_op[i][j]);
            y_weight += (temp * y_op[i][j]);
        } }
        edge = ABS(x_weight) + ABS(y_weight);
      }
      yc_out[index(row, col)] = edge;
    }
  }
}
```

Video 2: Run the Solution 1 on a Zynq board

- **1 FPS @ 1080p**
- **10x speedup**

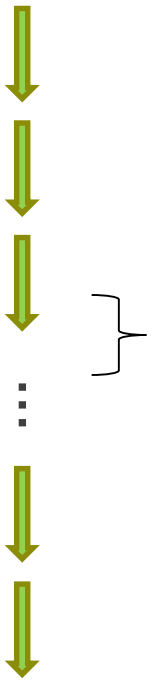
Problem 2: Loop Iterations Are Sequential

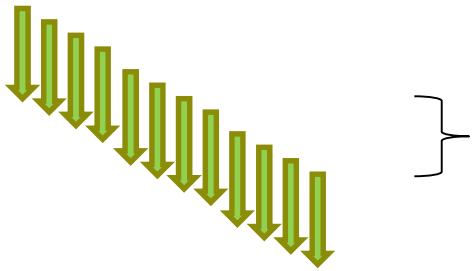


```
void sobel_filter(unsigned short *yc_in,unsigned short *yc_out, short *x_op, short *y_op)
{
  int row, col;
  for(row = 0; row < NUMROWS; row++){
    for(col = 0; col < NUMCOLS; col++){
      unsigned short input_data, unsigned char edge;
      if((col < NUMCOLS) & (row < NUMROWS))
        input_data = yc_in[row*NUMCOLS+col];
      if( isEdge(row, col)) edge=0;
      else{
        short x_weight = 0, y_weight = 0;
        for(char I = 0; i < 3; i++){
          for(char j = 0; j < 3; j++){
            unsigned short temp = (yc_in[index(row, col, I, j)]);
            x_weight += (temp * x_op[i][j]);
            y_weight += (temp * y_op[i][j]);
        } }
        edge = ABS(x_weight) + ABS(y_weight);
      }
      yc_out[index(row, col)] = edge;
    }
  }
}
```

Executing sequentially

Solution 2: Pipeline Loop Iterations





60*1920*1080 =

124,416,000 cycles

60+(1920*1080) =

2,073,660 cycles (60x speedup)

Assuming 60 cycles / loop iteration

Solution 2: Add Pipeline #pragma

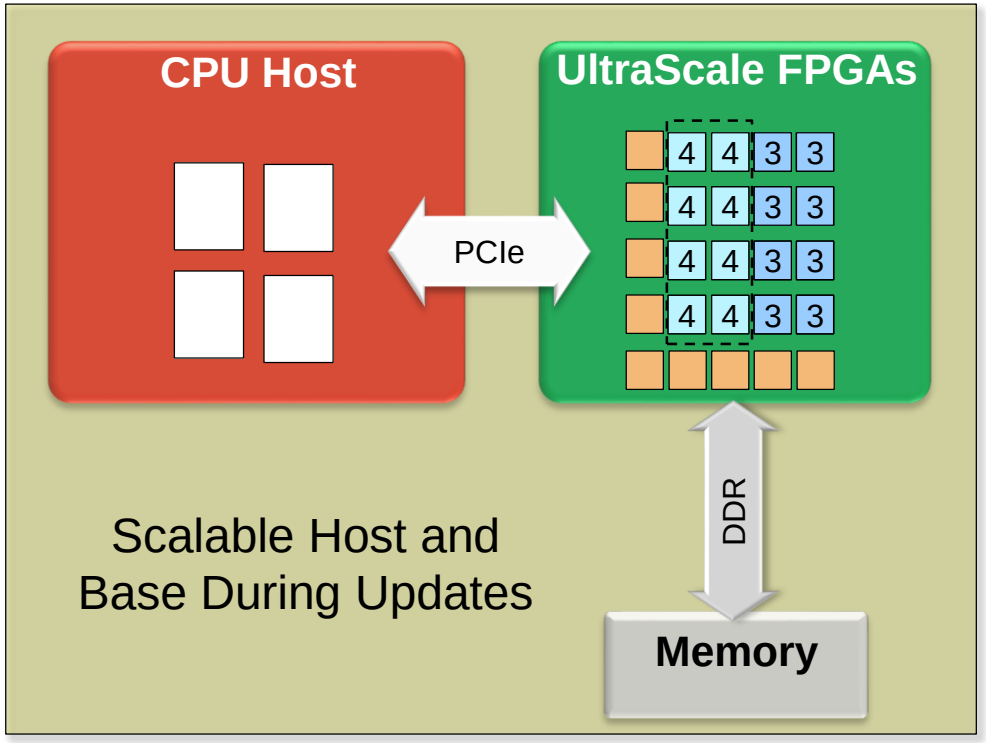
```
void sobel_filter(unsigned short *yc_in,unsigned short *yc_out, short *x_op, short *y_op)
{
  int row, col;
  ap_linebuffer<unsigned char, 3, NUMCOLS> buff_A;
  ap_window<unsigned char,3,3> buff_C;
  for(row = 0; row < NUMROWS; row++){
    for(col = 0; col < NUMCOLS; col++){
#pragma AP PIPELINE II = 1
      unsigned short input_data, unsigned char edge;
      update_linebuffer(buff_A, yc_in[index(row,col)]);
      update_window(buff_C);
      if((col < NUMCOLS) & (row < NUMROWS))
        input_data = yc_in[row*NUMCOLS+col];
      if( isEdge(row, col)) edge=0;
      else{
        short x_weight = 0, y_weight = 0;
        for(char I = 0; i < 3; i++){
          for(char j = 0; j < 3; j++){
            unsigned short temp = (buff_C[index(row, col, I, j)]);
            x_weight += (temp * x_op[i][j]);
            y_weight += (temp * y_op[i][j]);
        } }
        edge = ABS(x_weight) + ABS(y_weight);
      }
      yc_out[index(row, col)] = edge;
    }
  }
}
```

Video 3: Run the Solution 2 on a Zynq board

- **60 FPS @ 1080p**
- **60x speedup**

SDAccel

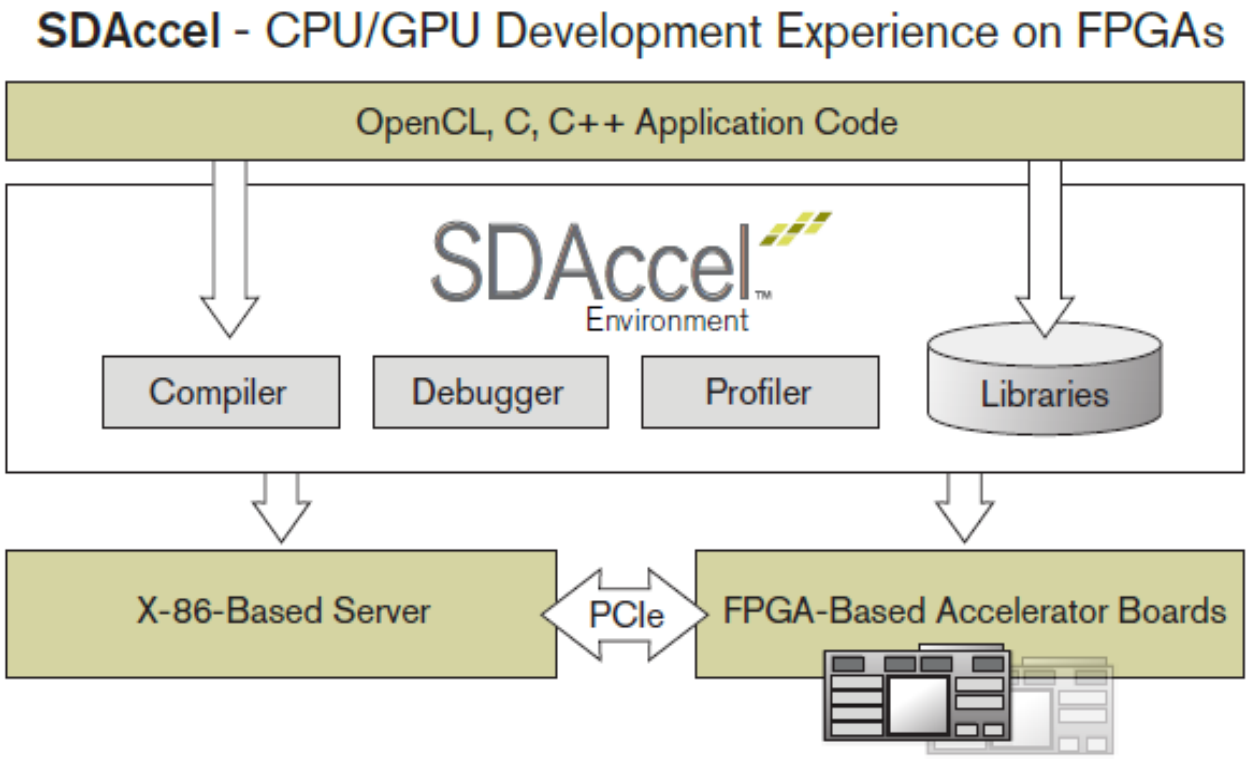
CPU/GPU-like Runtime Experience on FPGAs



CPU/GPU Runtime Experience

- On-demand loadable acceleration units
- Always on interfaces (Memory, Ethernet PCIe, Video)
- Optimize resources thru hardware reuse

SDAccel: Development Environment for C, C++ and OpenCL



Libraries	Availability
OpenCL built-ins	Included
Video, DSP, Linear Algebra	Included
OpenCV, BLAS	Provided by Auviz Systems

Readily Available Boards

CONVEY COMPUTER™ ALPHA DATA picocomputing

Typical deployment workflow

- **Optimize on x86 platform with emulator and auto generated cycle accurate models**
  - Identify application for acceleration
  - Program and optimize kernel on host
  - Compile and execute application for CPU
  - Estimate performance
  - Debug FPGA kernels with cycle accurate models on CPU
- **Deployment on FPGA**
  - Compile for FPGA (longest step)
  - Execute and validate performance on card

Implementation

- **Each OpenCL work-group mapped on 1 IP block**
- **Pipelining of work-items in a work-group possible with #pragma**
- **Support several kernels per program**
- **Generate FPGA container with bitstream + metadata**
- **Different compilation/execution modes**
  - CPU-only mode for host and kernels
  - RTL version of kernels for co-simulation
  - Real FPGA execution
- **Estimation of resource utilization by the tool**
- **OpenCL host API handles accelerator invocation**

Specify work-group size for better implementation

```
__kernel
__attribute__((reqd_work_group_size(4,4,1)))
void mmult32(__global int* A, __global int* B,
__global int* C) {
  // 2D Thread ID
  int i = get_local_id(0);
  int j = get_local_id(1);
  __local int Blocal[256];
  int result=0, k=0;
  B_local[i*16 + j] = B[i*16 + j];
  barrier(CLK_LOCAL_MEM_FENCE);
  for(k = 0; k < 16; k++)
    result += A[i*16 + k]*B_local[k*16 + j];
  C[i*16 + j] = result;
}
```

- **Is compiled into**

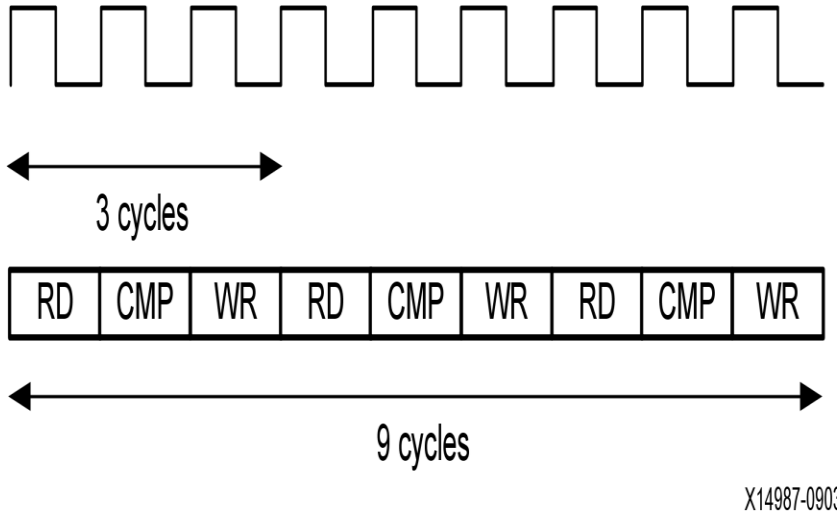
```
__kernel void mmult32(global int* A, global int* B,
global int* C) {
  localid_t id;
  int B_local[16*16];
  for(id[2] = 0; id[2] < 1; id[2]++)
    for(id[1] = 0; id[1] < 4; id[1]++)
      for(id[0] = 0; id[0] < 4; id[0]++) {
        ...
      }
  ...
}
```

Loop unrolling

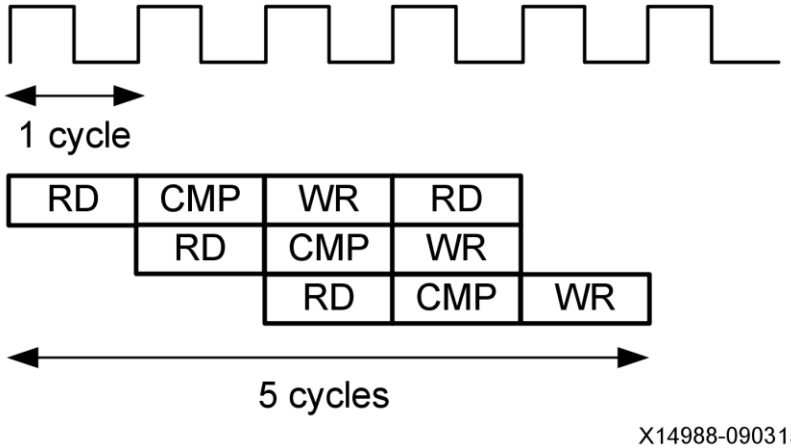
```
kernel void
vmult(local int* a, local int* b, local int* c) {
  int tid = get_global_id(0);
  __attribute__((opencl_unroll_hint(2)))
  for (int i = 0; i < 4; i++) {
    int idx = tid*4 + i;
    a[idx] = b[idx]*c[idx];
  }
}
```

Work-item pipelining

```
__attribute__ ((reqd_work_group_size(3,1,1)))
kernel void foo(...) {
  int tid = get_global_id(0);
  op_Read(tid);
  op_Compute(tid);
  op_Write(tid);
}
```




```
__attribute__ ((reqd_work_group_size(3,1,1)))
kernel void foo(...) {
  __attribute__((xcl_pipeline_workitems)) {
    int tid = get_global_id(0);
    op_Read(tid);
    op_Compute(tid);
    op_Write(tid);
  }
}
```



Partitioning memories inside of compute units

- **Useful to avoid access conflict**
- **__local int buffer[16] __attribute__((xcl_array_partition(cyclic,4,1)));**
  - Cyclic partition on 4 memories along dimension 1
- **___local int buffer[16] __attribute__((xcl_array_partition(block,4,1)));**
  - Bloc partition on 4 memories along dimension 1
- **__local int buffer[16] __attribute__((xcl_array_partition(complete,1)));**
  - Complete partitioning along dimension 1: 1 register/element ☺

Using HLS C/C++ as OpenCL kernel

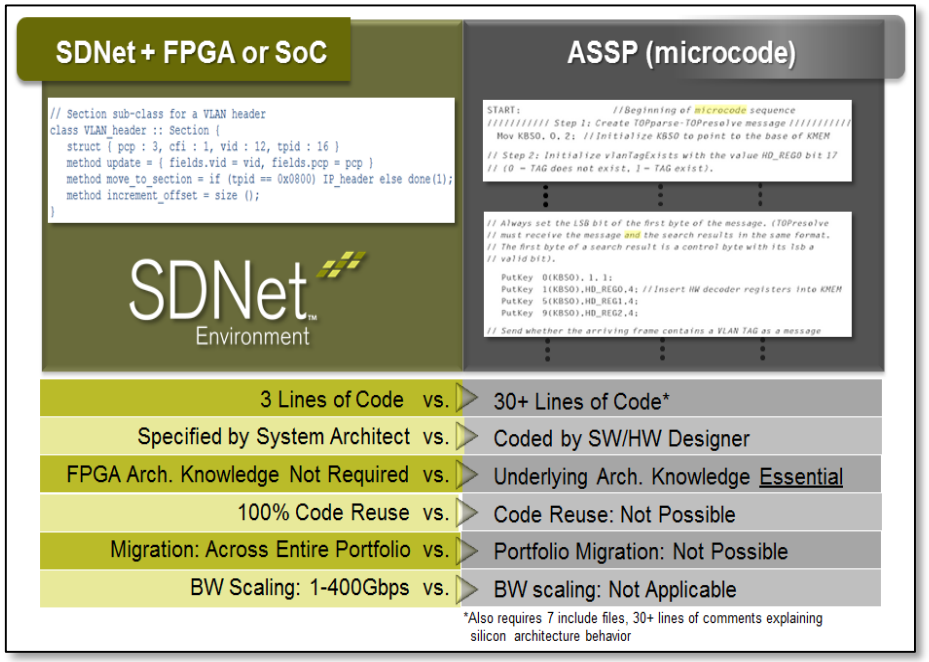
- Need to use parameter interface compatible with SDAccel OpenCL

```
void matrix_multiplication(int *a, int *b, int *output) {
#pragma HLS INTERFACE m_axi port=a offset=slave bundle=gmem
#pragma HLS INTERFACE m_axi port=b offset=slave bundle=gmem
#pragma HLS INTERFACE m_axi port=output offset=slave bundle=gmem
#pragma HLS INTERFACE s_axilite port=a bundle=control
#pragma HLS INTERFACE s_axilite port=b bundle=control
#pragma HLS INTERFACE s_axilite port=output bundle=control
#pragma HLS INTERFACE s_axilite port=return bundle=control
  // Matrices of size 16*16
  const int rank = 16;
  int sum = 0;
  // Cache the external matrices
  int bufa[rank*rank];
  int bufb[rank*rank];
  int bufc[rank*rank];
  memcpy(bufa, a, sizeof(bufa));
  memcpy(bufb, b, sizeof(bufb));
  for (unsigned int c = 0; c < rank; c++) {
    for (unsigned int r = 0; r < rank; r++) {
      sum = 0;
      for (int index = 0; index < rank; index++) {
#pragma HLS pipeline
        int aIndex = r*rank + index;
        int bIndex = index*rank + c;
        sum += bufa[aIndex]*bufb[bIndex];
      }
      bufc[r*rank + c] = sum;
    }
  }
  memcpy(output, bufc, sizeof(bufc));
  return;
}
```

SDNet

SDNet: SW Defined Specification Environment

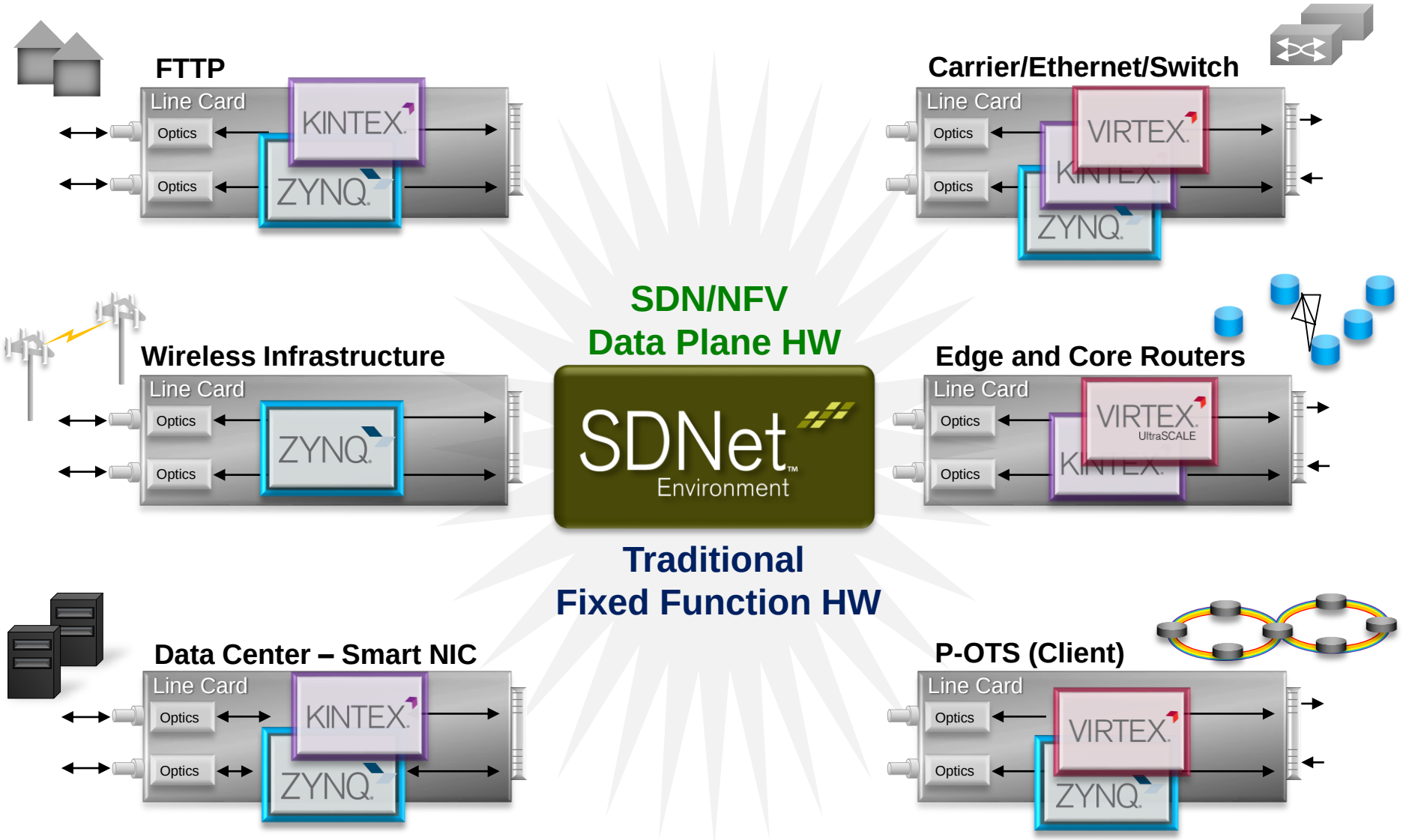
Replaces NPUs with All Programmable performance, flexibility, and security



10x productivity advantage

- **'Softly Defined' networks**
  - Content intelligence data plane hardware supporting hit-less in service updates
  - Dynamically collaborates with control plane
  - Address performance, flexibility, and security challenges of agile service-oriented networking
- **SDNet's specification driven environment**
  - Generates packet data plane HW subsystem
  - Generates subsystem firmware
  - Generates optimized packet processing engines and flows
    - e.g. parsing, editing, search, and feature optimized QoS policy
  - Generates test benches

Data Plane Function Acceleration – Core to Edge



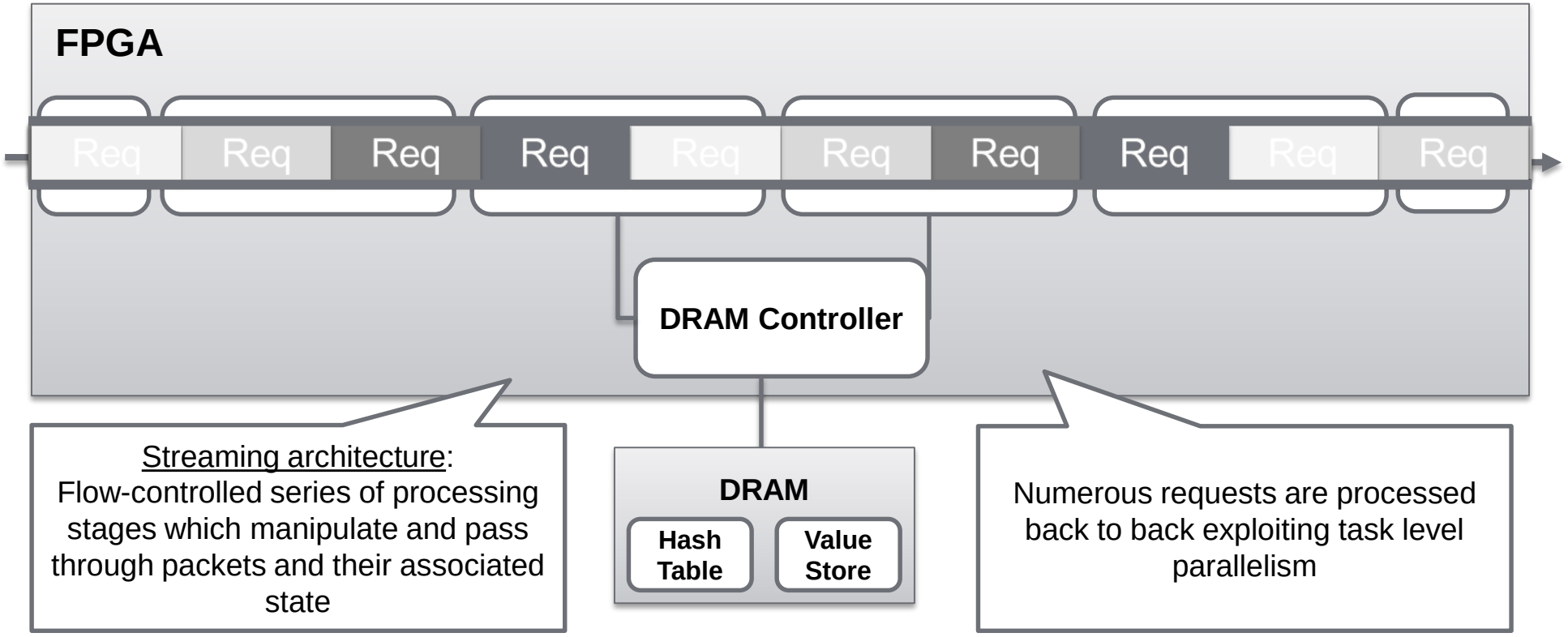
Application example

Memcached

- **Common middleware application to alleviate access bottlenecks on databases**
  - Most popular and most recent database contents are cached in main memory of a tier of server platforms
  - Provides the abstraction of an associative memory
    - Values are stored or retrieved by sending the associated key

- **Demonstrated performance scalability using custom dataflow architectures (measured 35x performance/power)**

Dataflow architectures to scale performance

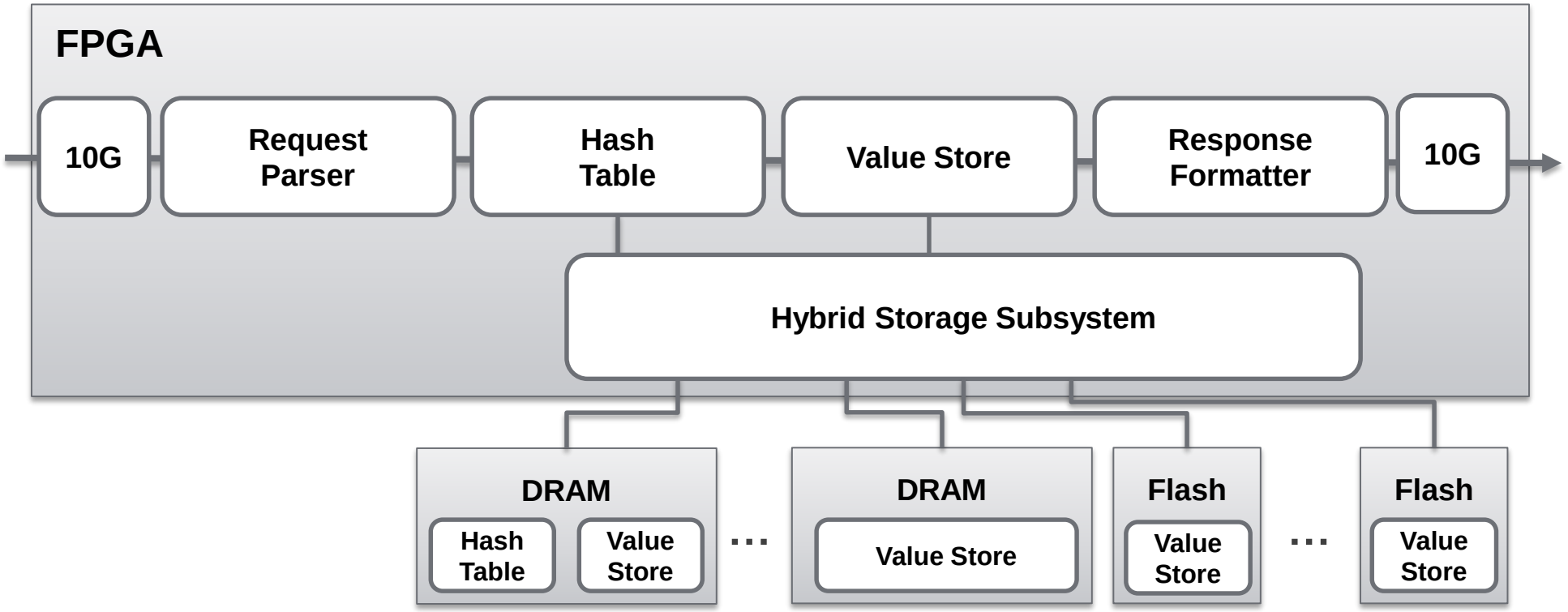


- **10Gbps demonstrated with a 64b data path @ 156MHz using 3% of FPGA resources**
- **80Gbps can be achieved by using a 512b @ 156MHz pipeline for example**

Source: [4] Blott et al: Achieving 10Gbps line-rate key-value stores with FPGAs; HotCloud 2013

How can we scale storage capacity with flash without sacrificing performance?

- **Lower in cost, lower in power and higher in capacity**



Object distribution on the basis of size

Value Size (B)	128	256	512	768	1K	4K	8K	32K	1M
Facebook	0.55	0.075	0.275	0	0	0	0	0	0.1
Twitter	0	0	0	0.1	0.85	0.05	0	0	0
Wiki	0	0	0.2	0.1	0.4	0.29	0.008	0.001	0.001
Flickr	0	0	0	0	0	0.9	0.05	0.03	0.02

Stored in DRAM

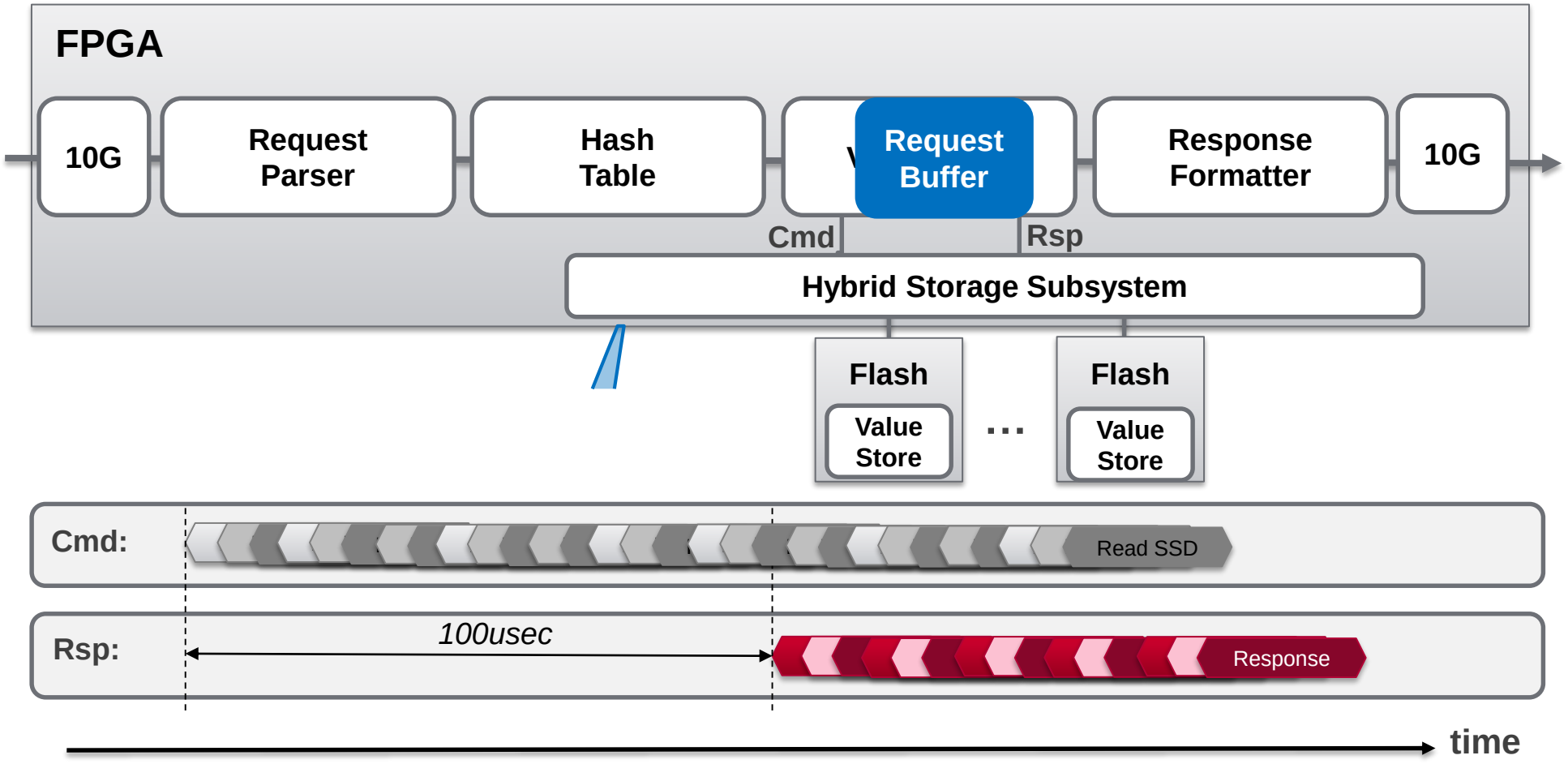
Stored in Flash

- **Advantages:**
  - Larger objects require larger storage
  - Larger granular access to flash suits page-size access granularity of flash
- **Concerns:**
  - Large access latency on flash
  - Variations in access bandwidth and latency between DRAM and flash

Source: *[3] Atikoglu et al: Workload analysis of a large-scale key-value store; SIGMETRICS 2012*

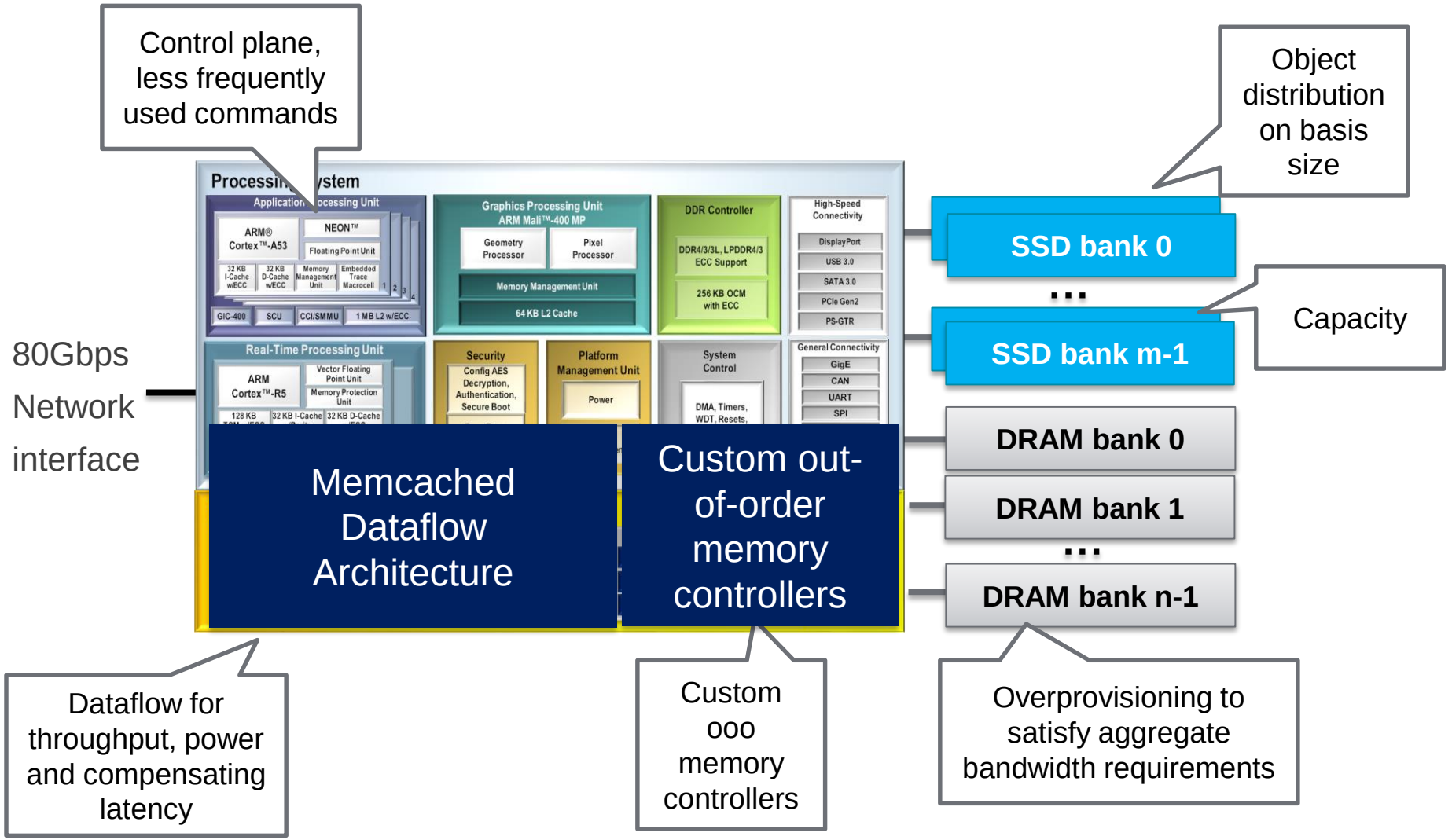
[13] Lim et al: Thin servers with smart pipes: designing {SoC} accelerators for memcached; ISCA 2013

Dataflow architectures can accommodate high latency accesses without sacrificing throughput



- In dataflow architectures: no limit to number of outstanding requests
- Flash can be serviced at maximum speed

Proposed single node system architecture



Conclusion

- **Modern FPGAs are complex MP-SoC with programmable logic, CPU, GPU, peripherals, IO… the ultimate accelerator**
- **Require different abstractions to program with productivity at different levels**
- **SDSoC to ease system-level programming**
- **SDAccel: OpenCL for host-accelerator style programming**
  - Add specific optimizations for FPGA power & efficiency tradeoffs
    - data sizes
    - #CU
    - Pipelines
    - Bus
    - Memories
    - Pipes
    - Specific interfaces & IO…
- **SDNet: DSL for software defined network**
- **What's next? For example the great unification with SYCL C++ (talk tomorrow morning)…**