CORE | Vision
NEXT LEVEL | EMBEDDED DEVELOPMENT

FPGA hardware acceleration turns out to be a software based design flow

Frank de Bont
Trainer / Consultant

Cereslaan 10b
5384 VT Heesch
+31 (0)412 660088
info@core-vision.nl
www.core-vision.nl

Accelerators and Systems

- An accelerator is a dedicated piece of IP implemented in the configurable logic of an SoC and coupled to the processing system
- The goal is to offload the processor's computationally intensive tasks to the hardware where it can be executed at a significantly higher rate
- The design of the internals of the accelerator is referred to as the microarchitecture and is governed by coding style and #pragmas

System Design Challenges

- **How to connect the processor to the accelerator?**
  - AXI ports: general-purpose masters and slaves, ACP, high performance, ACE, HPC
  - Interrupts, WFE, WFI, polling
  - Clocking
  - Cache and memory utilization
  - Data movement (DMA, datamover)

- **How to coordinate hardware and software?**
  - Polling versus interrupting
  - Knowing when the DMA and accelerator(s) are done
  - Knowing where the data is at the end of an acceleration process
  - Blocking versus non-blocking coding styles and support

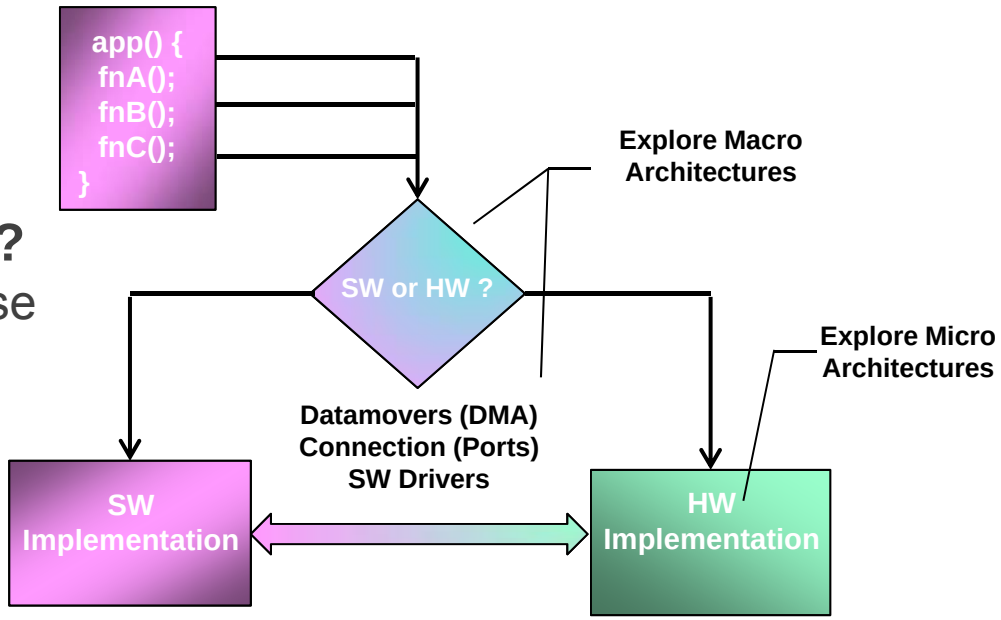
General Goals of a PL-based Accelerator

- **Achieving higher computing performance this is the primary objective**
- **Saving processor cycles by offloading the computation**
- **High performance of the PL-based accelerator itself**
  - Lower latency
  - Higher throughput
  - Several times faster compared to software-based computation

- **Ensure that data transfer delays between PS and accelerator do not eliminate the performance gain from the accelerator**

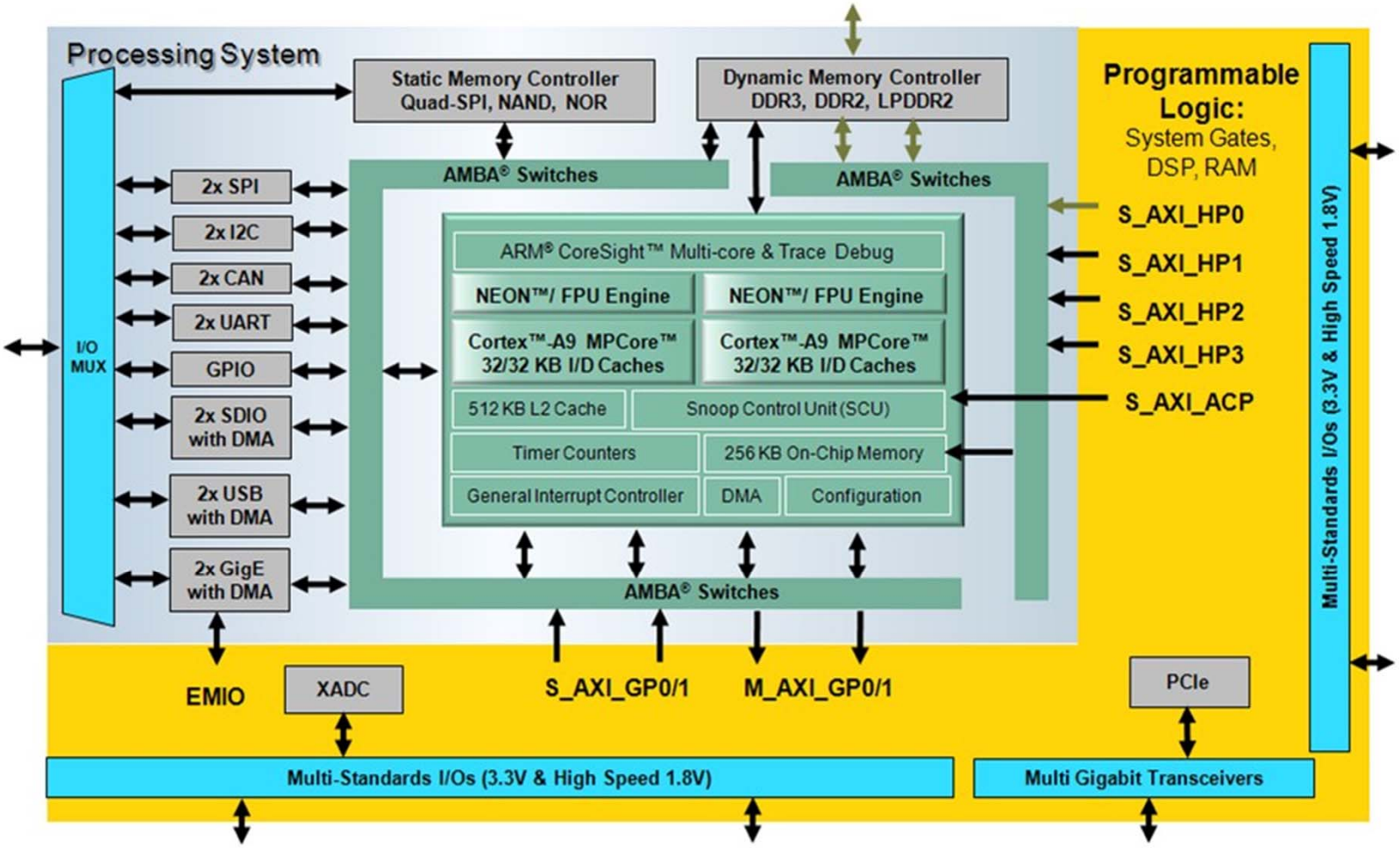
System-level Considerations

- **What gets accelerated ?**
- **How is software implemented in hardware?**
  - Is hardware design epertise available?
- **How will software and hardware talk to each other?**



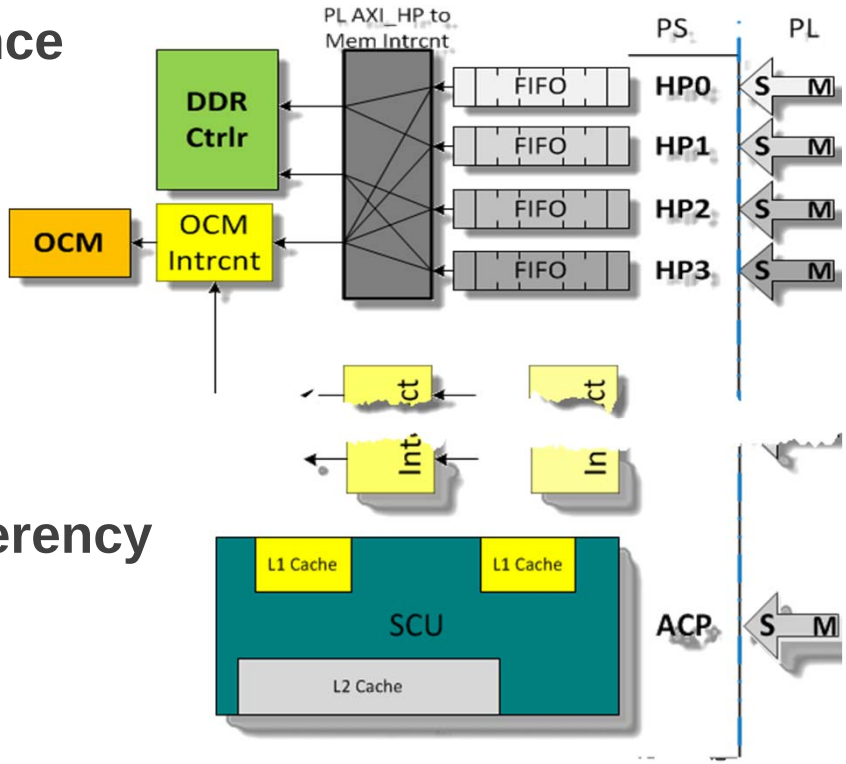
- **Will it meet performance requirements the first try?**
  - What changes are required at the macro/micro-architecture levels (or both)?

Zynq-7000 SoC Block Diagram

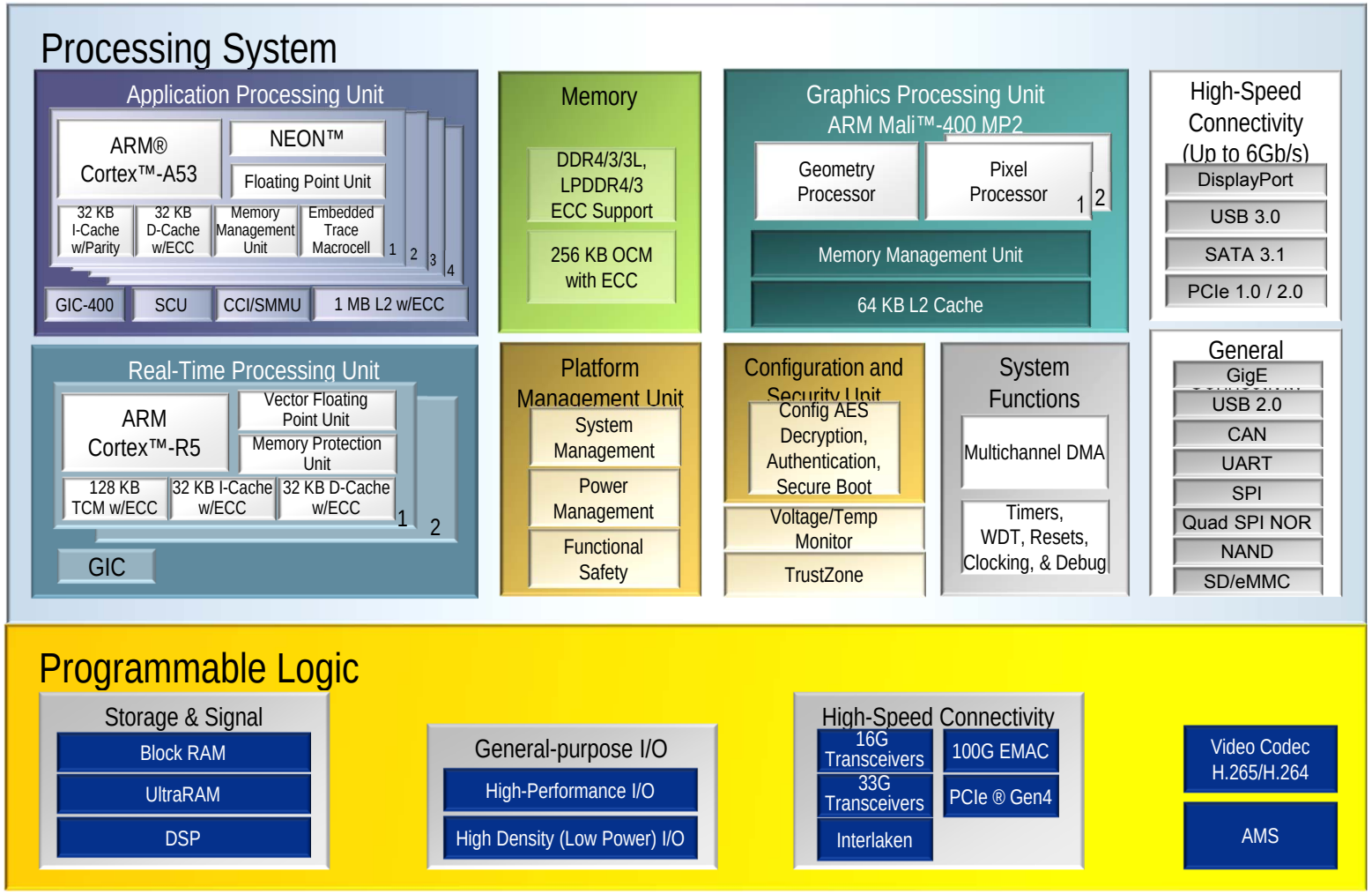


Zynq Accelerator Interfaces

- **Four AXI High-Performance slave ports**
  - S_AXI_HP0
  - S_AXI_HP1
  - S_AXI_HP2
  - S_AXI_HP3
- **One AXI accelerator coherency slave port**
  - S_AXI_ACP

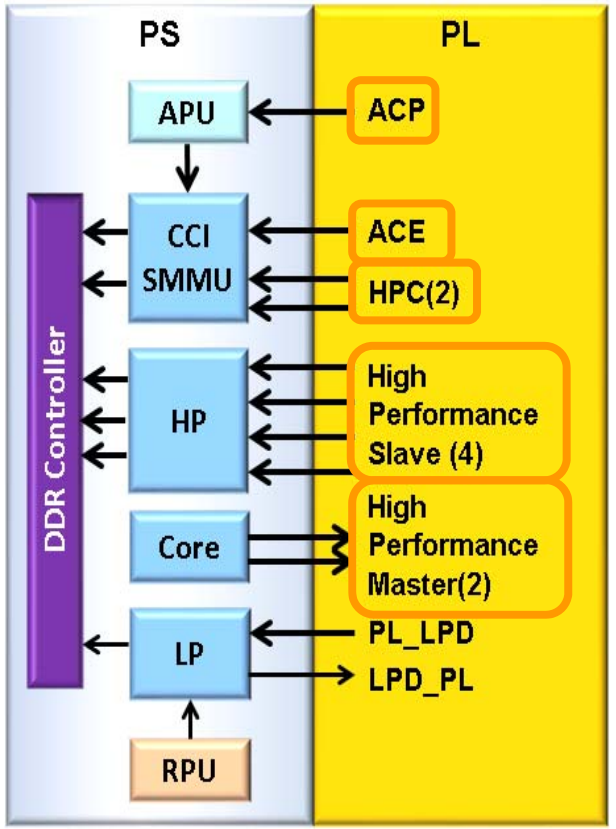


Zynq UltraScale+ MPSoC



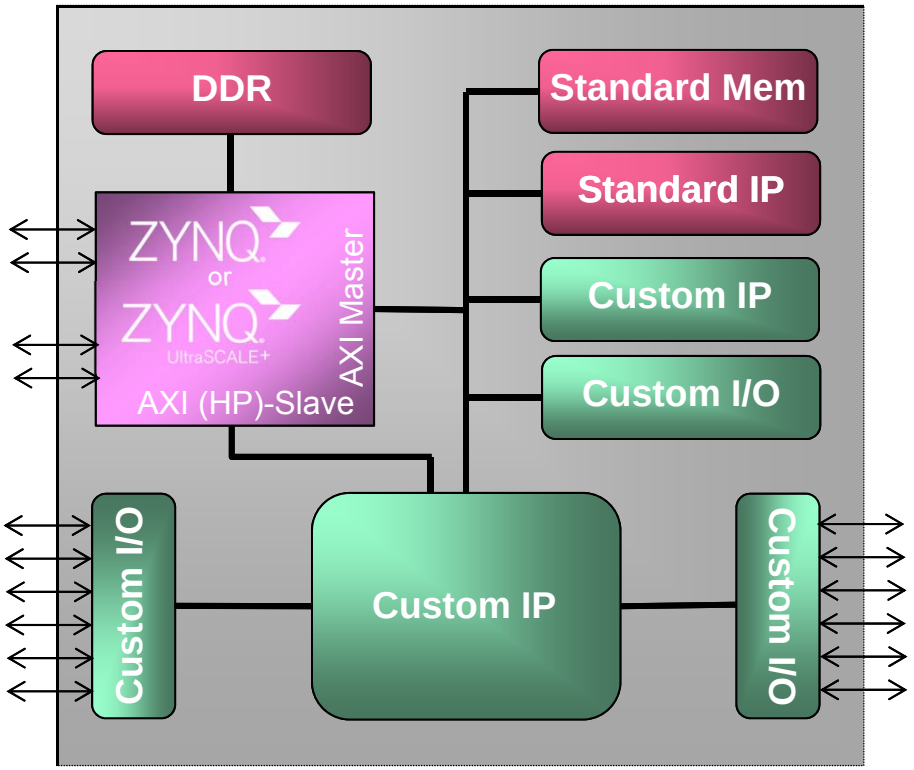
Zynq UltraScale+ Accelerator Interfaces

- Accelerator coherency port ACP
- AXI coherency extension ACE
- Two High-Performance coherency interfaces HPC
- Four AXI High-Performance slave ports
- Two High-Performance master ports
  - Can be accessed from APU or RPU

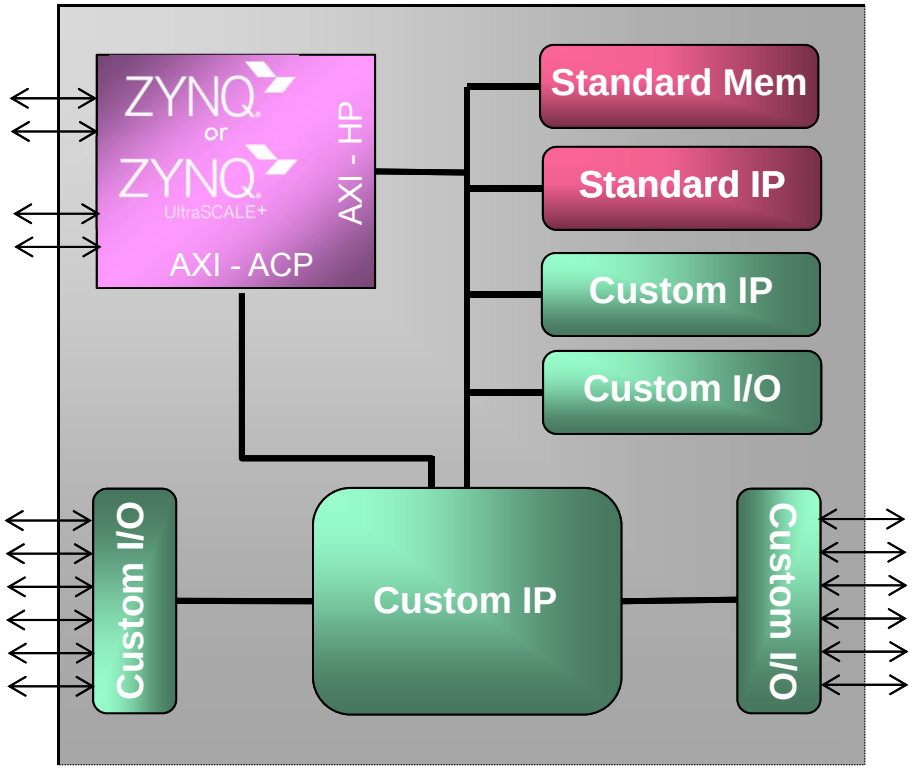


Data Flow Model

- **Custom IP for complex function and data flow**
- **PS used for control and resource management**
  - Minimal to no data processing by the CPUs
- **Custom IP in PL operates nearly autonomously from the PS**
  - May play through to acccess the DDR using the HP ports



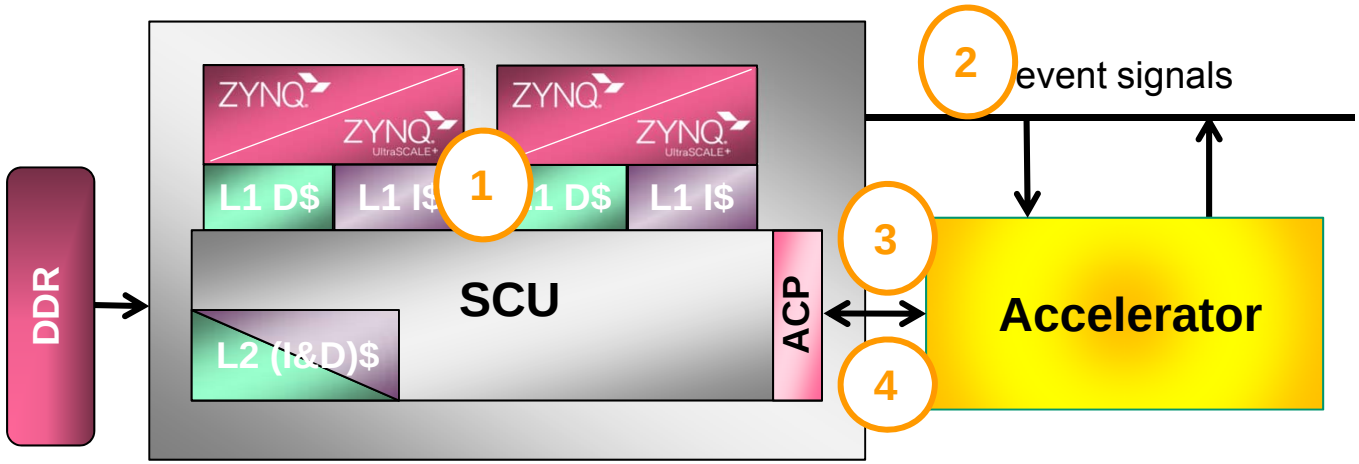
Acceleration Model



- **PS primary configures data for the accelerator**
  - Can also perform significant tasks
- **PL for hardware acceleration**
  - Custom IP tightly coupled with processor
  - Accelerator reacts to PS
- **Communications between**
  - GP ports uses for accelerator management
  - Data moved on high-efficiency ports (ACP/HPx)
  - Interrupts or event signals used to signal significant occurrences

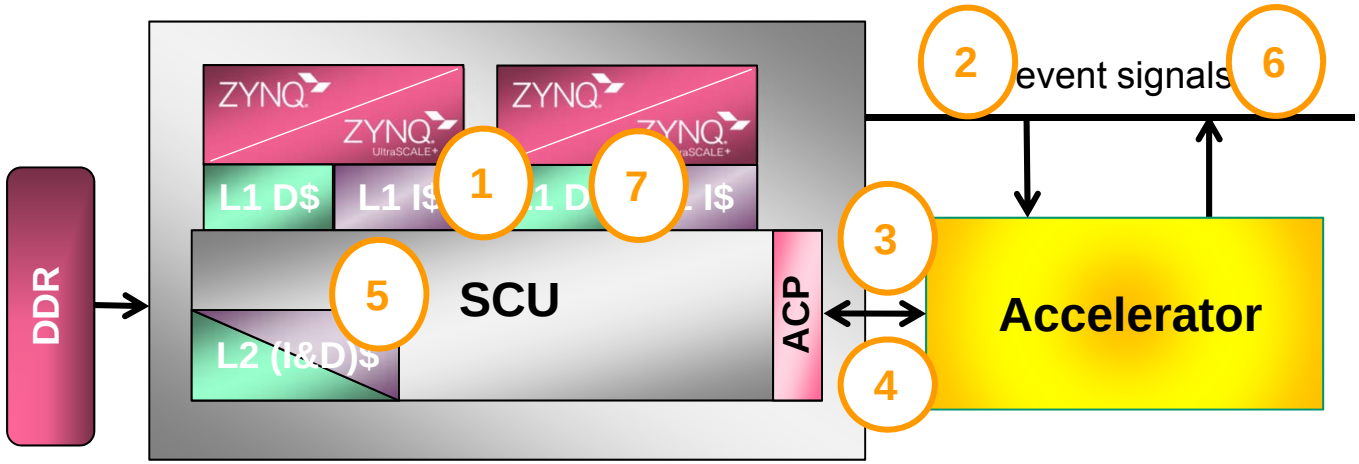
Typical ACP Accelerator Example

- 1. CPU leaves (updates) data in either the L1 or L2 cache depending on the volume of data to move to the accelerator.
- 2. CPU notifies the accelerator via the event bus to begin data operations.
- 3. The Accelerator issues are read into the SCU via an AXI slave through the ACP. Data may be returned from L1 or L2 cache, OCM, or (worst case) from DDR.
- 4. After processing, the accelerator writes back into the specified memory location which may be in L1, L2, OCM, or DDR via the AXI slave connected to the ACP.

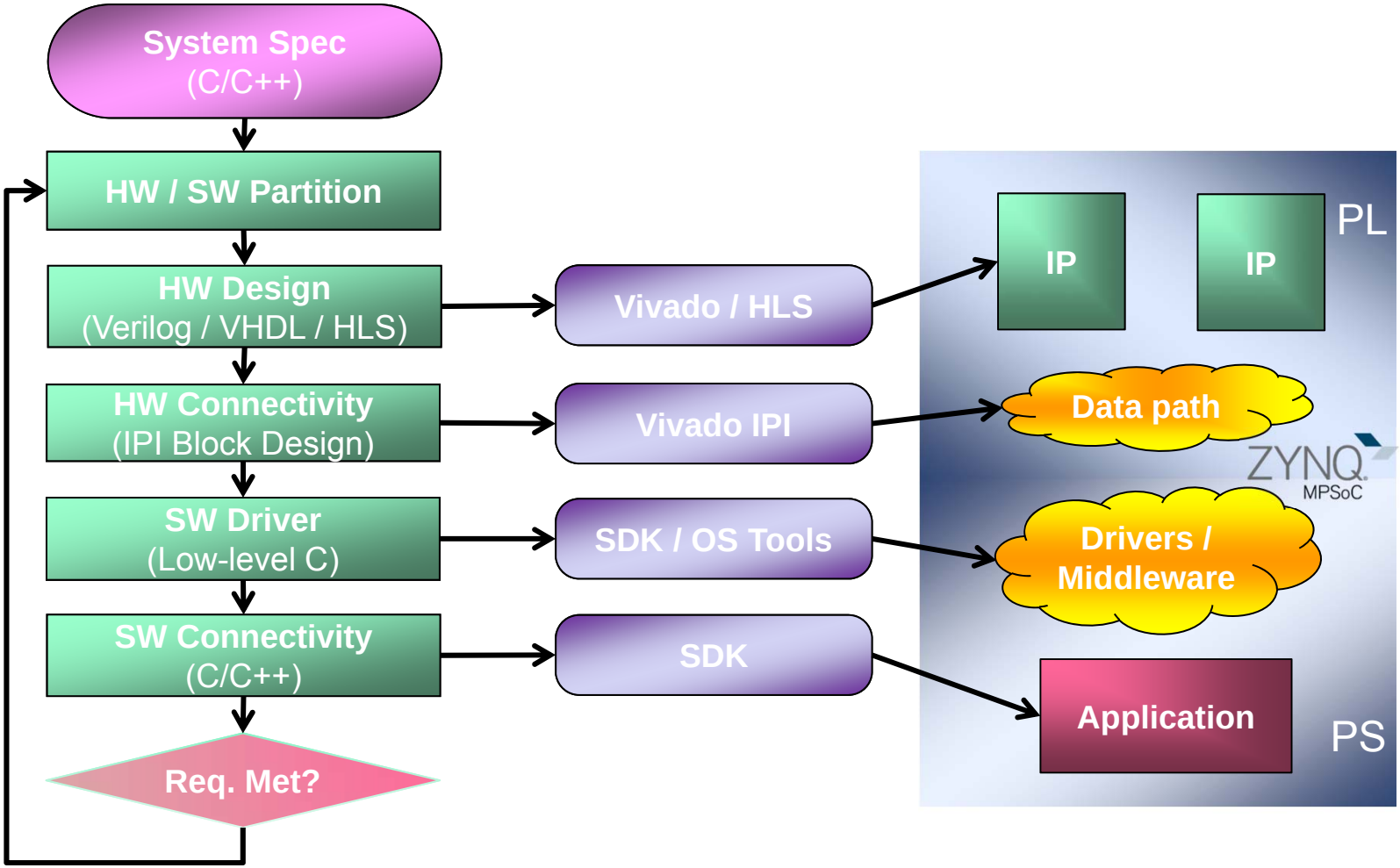


Typical ACP Accelerator Example cont

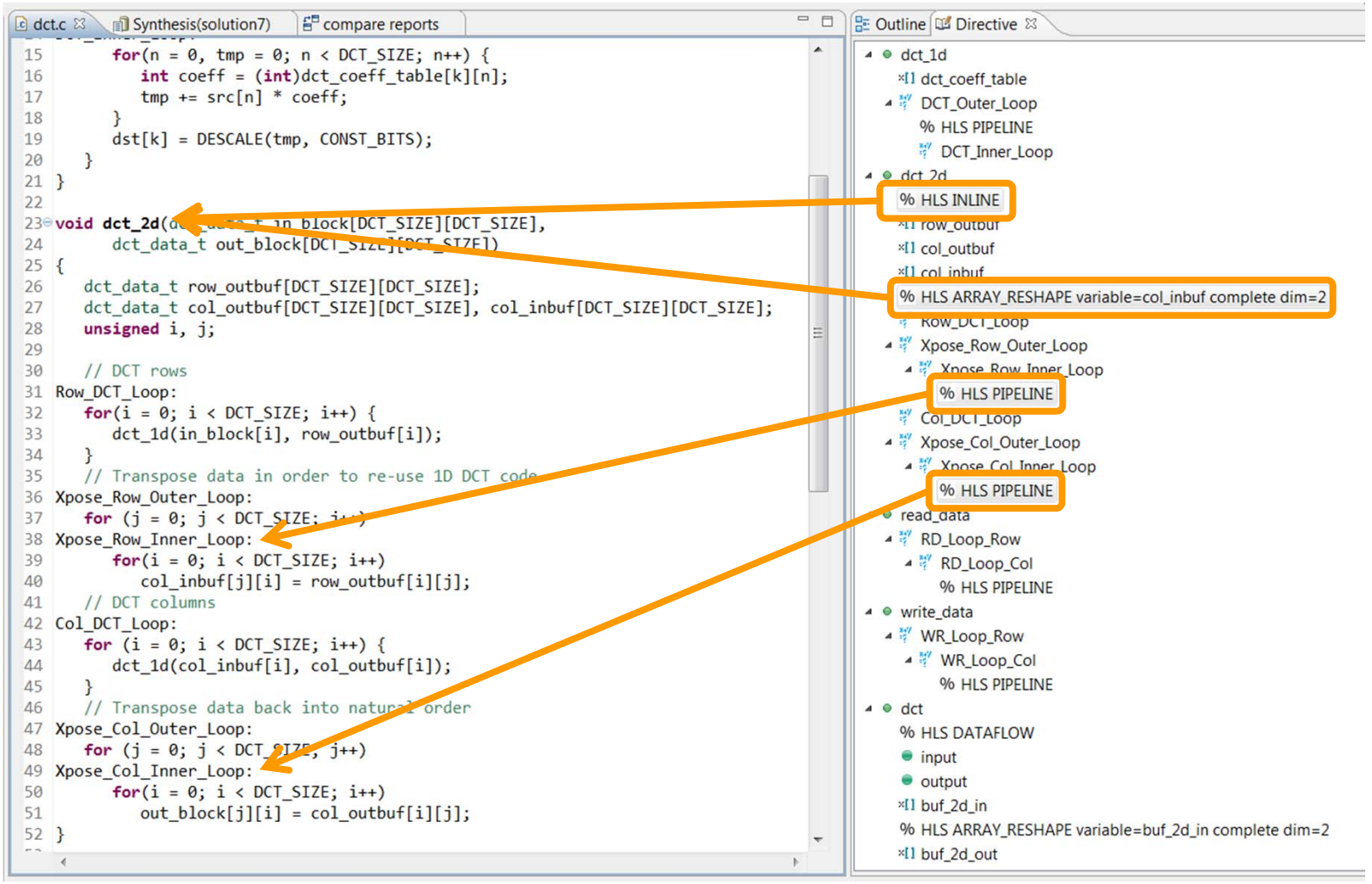
- 5. The SCU ensures coherency by placing the data into the appropriate location, ideally L1 or L2 cache, but may be into the DDR. This is handled transparently by the SCU; neither the accelerator nor the CPUs need to worry about this.
- 6. The Accelerator notifies the PS via the event bus that it has completed.
- 7. The Accelerator is now out of the picture and one or both of the CPUs begin operating on the returned data which should now be in a near (fast) memory (L1, L2, OCM). Where there is too much data or the wrong addresses are targeted, data movement will involve DDR or other slower memories.



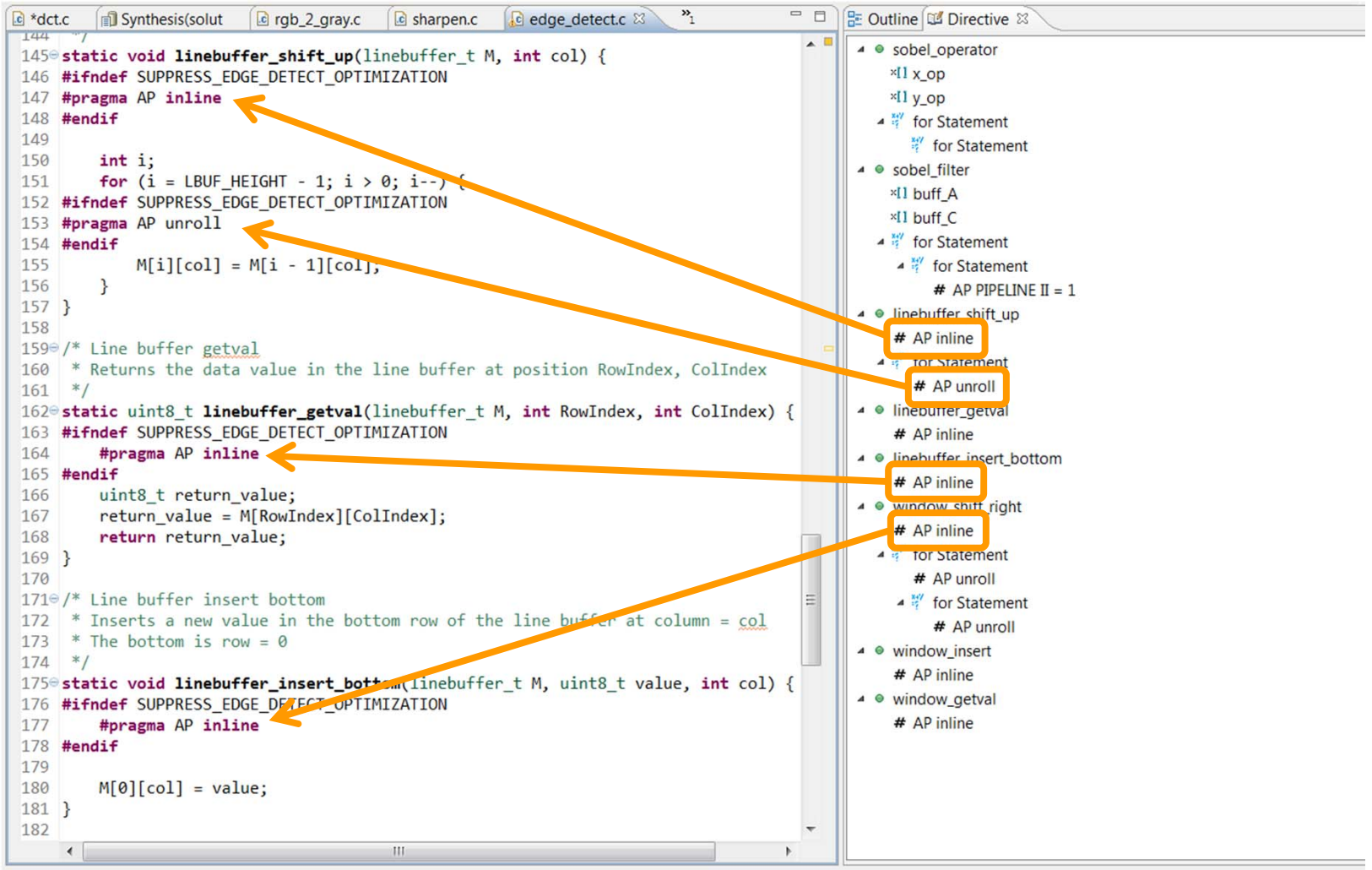
Design Flow without SDSoC



Add Directives to your C/C++-code



Add #Pragma to your C/C++-code




```
144 */
145 static void linebuffer_shift_up(linebuffer_t M, int col) {
146 #ifndef SUPPRESS_EDGE_DETECT_OPTIMIZATION
147 #pragma AP inline
148 #endif
149
150     int i;
151     for (i = LBUF_HEIGHT - 1; i > 0; i--) {
152 #ifndef SUPPRESS_EDGE_DETECT_OPTIMIZATION
153 #pragma AP unroll
154 #endif
155         M[i][col] = M[i - 1][col];
156     }
157 }
158
159 /* Line buffer getval
160  * Returns the data value in the line buffer at position RowIndex, ColIndex
161  */
162 static uint8_t linebuffer_getval(linebuffer_t M, int RowIndex, int ColIndex) {
163 #ifndef SUPPRESS_EDGE_DETECT_OPTIMIZATION
164     #pragma AP inline
165 #endif
166     uint8_t return_value;
167     return_value = M[RowIndex][ColIndex];
168     return return_value;
169 }
170
171 /* Line buffer insert bottom
172  * Inserts a new value in the bottom row of the line buffer at column = col
173  * The bottom is row = 0
174  */
175 static void linebuffer_insert_bottom(linebuffer_t M, uint8_t value, int col) {
176 #ifndef SUPPRESS_EDGE_DETECT_OPTIMIZATION
177     #pragma AP inline
178 #endif
179
180     M[0][col] = value;
181 }
182
```

Compare different Solutions

- **Each solution uses a different directive file**
  - Constraints
- **Improved latency using a pipeline directive or #pragma**
- **Performance gain comes with area overhead**

dct.c | Synthesis(solution7) | compare reports

Vivado HLS Report Comparison

All Compared Solutions

solution7: xc7z020clg484-1

solution1: xc7z020clg484-1

Performance Estimates

Timing (ns)

Clock		solution7	solution1
ap_clk	Target	10.00	10.00
	Estimated	9.73	6.60

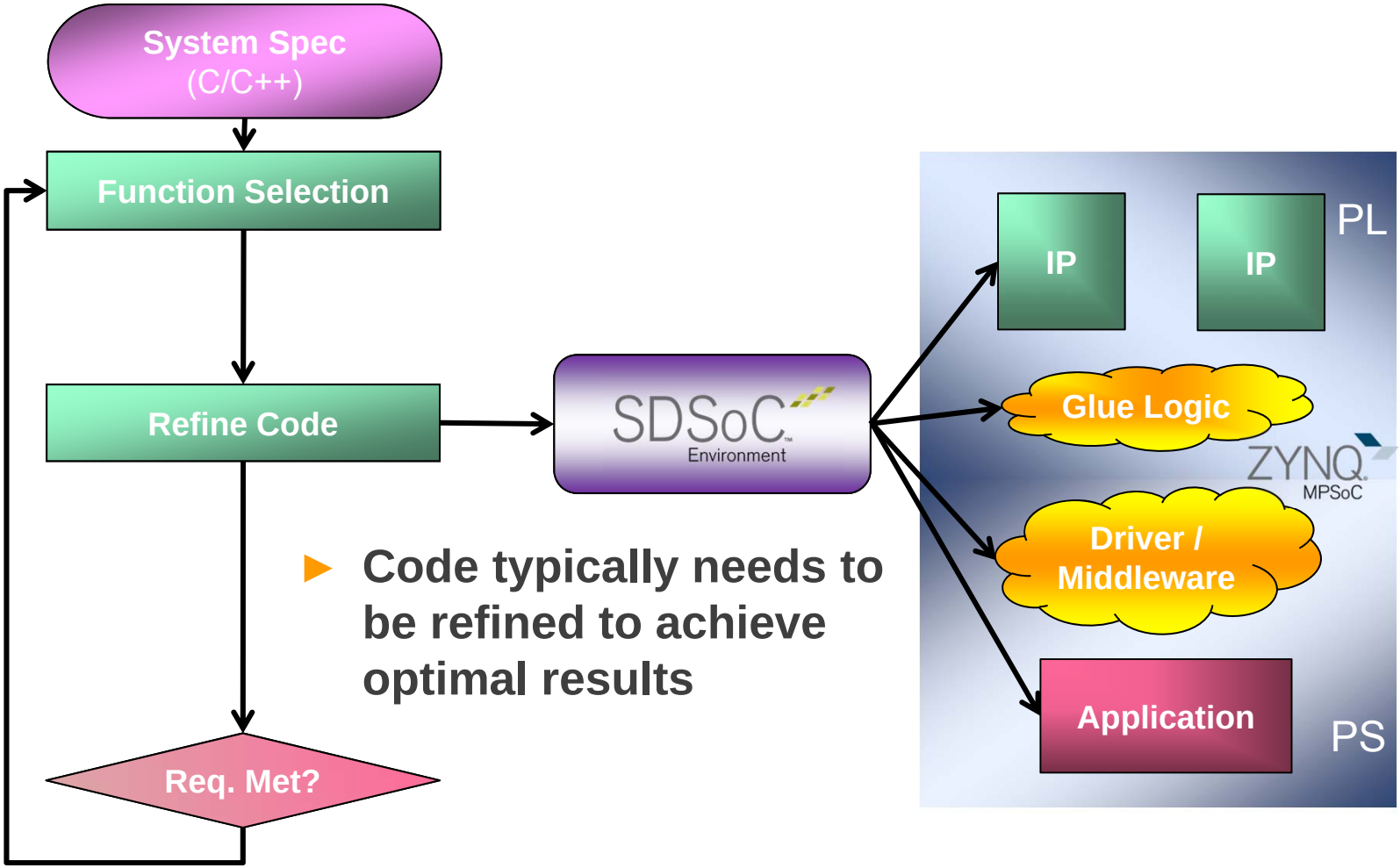
Latency (clock cycles)

		solution7	solution1
Latency	min	627	3959
	max	627	3959
Interval	min	132	3960
	max	132	3960

Utilization Estimates

	solution7	solution1
BRAM_18K	22	5
DSP48E	16	1
FF	3769	272
LUT	5499	874

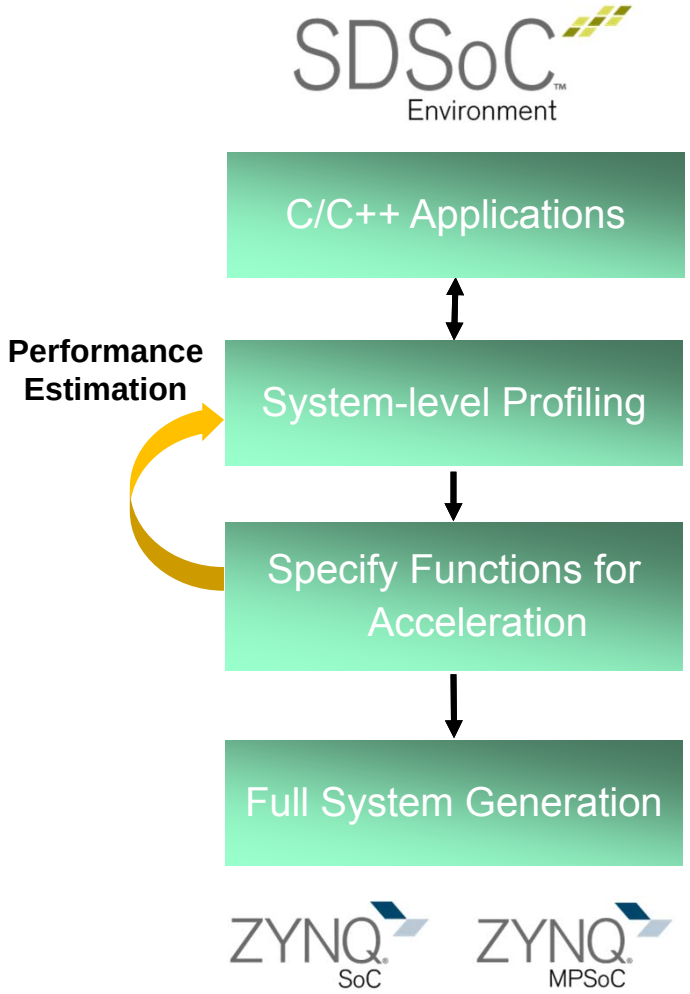
Design Flow with SDSoC



- Code typically needs to be refined to achieve optimal results

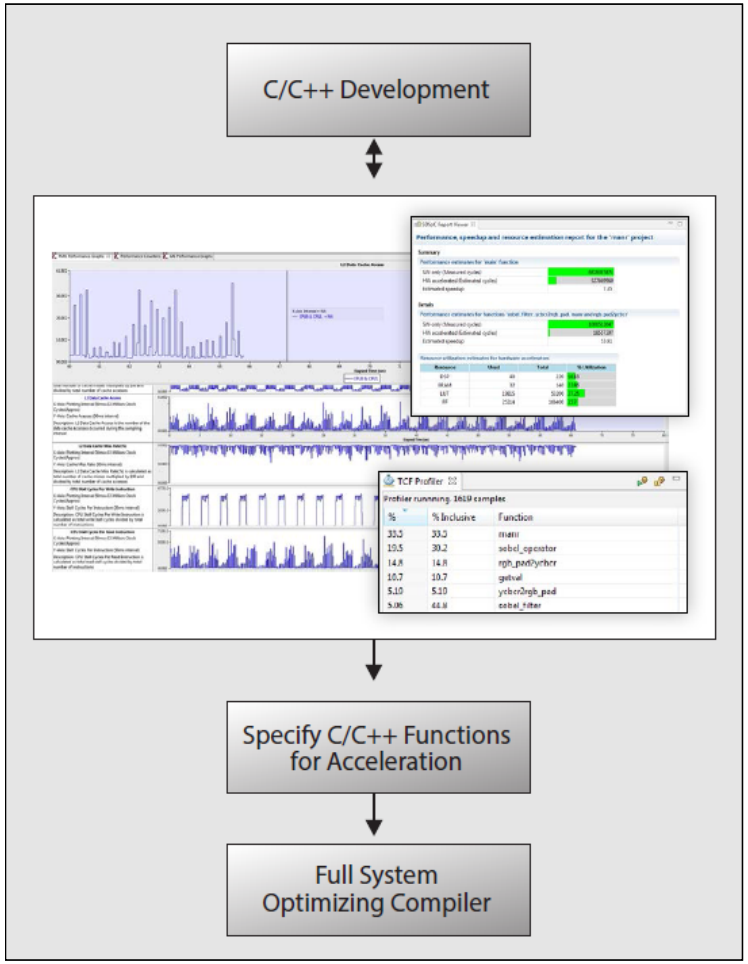
Embedded Design Flow with SDSoC

- Migrate C/C++ functions to hardware
- System-level debug and profile
- Simple hardware-software partitioning
- Full system generation including driver and hardware connectivity

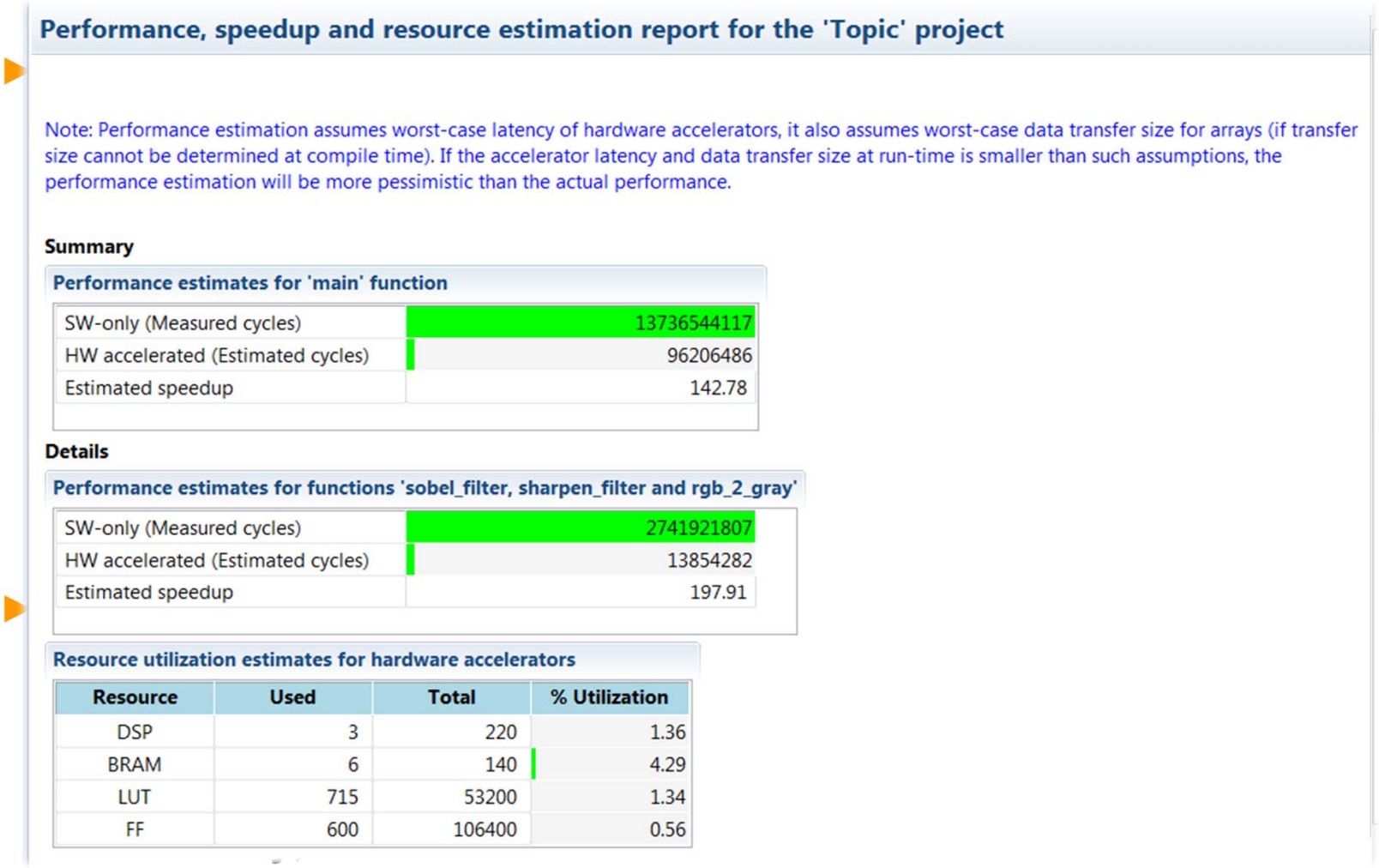


SDSoC System Level Profiling

- **Rapid system performance estimation**
  - **Full system estimation (programmable logic, data communication, processing system)**
  - **Reports SW/HW cycle level performance and hardware utilization**
- **Automated performance measurement**
  - **Runtime measurement by instrumentation of cache, memory, and bus utilization**



SDSoC System Level Profiling



Performance, speedup and resource estimation report for the 'Topic' project

Note: Performance estimation assumes worst-case latency of hardware accelerators, it also assumes worst-case data transfer size for arrays (if transfer size cannot be determined at compile time). If the accelerator latency and data transfer size at run-time is smaller than such assumptions, the performance estimation will be more pessimistic than the actual performance.

Summary

Performance estimates for 'main' function

SW-only (Measured cycles)	13736544117
HW accelerated (Estimated cycles)	96206486
Estimated speedup	142.78

Details

Performance estimates for functions 'sobel_filter, sharpen_filter and rgb_2_gray'

SW-only (Measured cycles)	2741921807
HW accelerated (Estimated cycles)	13854282
Estimated speedup	197.91

Resource utilization estimates for hardware accelerators

Resource	Used	Total	% Utilization
DSP	3	220	1.36
BRAM	6	140	4.29
LUT	715	53200	1.34
FF	600	106400	0.56

Core|Vision

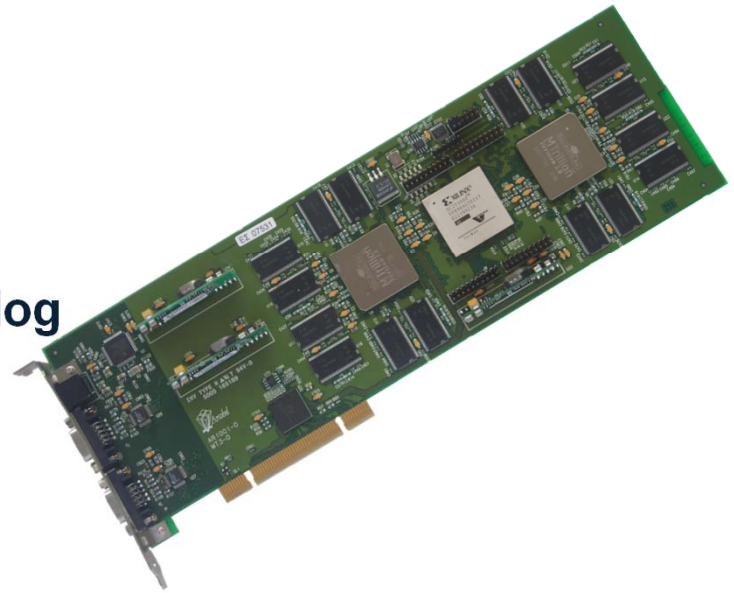
12 OKT BRABANTHALLEN DEN BOSCH D&E event 2017

XILINX ALL PROGRAMMABLE

Our competences

Core|Vision has more than 125 man years of design experience in hard- and software development. Our competence areas are:

- System Design
- FPGA Design
- Consultancy / Training
- Digital Signal Processing
- Embedded Real-time Software
- App development, IOS Android
- Data Acquisition, digital and analog
- Modeling & Simulation
- PCB design & Layout
- Doulos & Xilinx Training Partner



CORE | Vision
NEXT LEVEL | EMBEDDED DEVELOPMENT

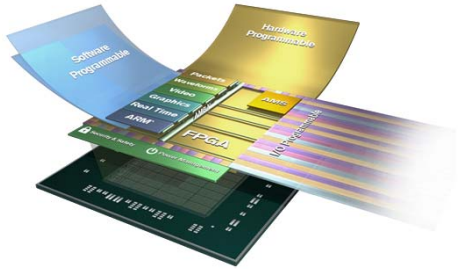
Q&A

Cereslaan 10b
5384 VT Heesch
✆ +31 (0)412 660088

www.*core-vision*.nl
Email : info@*core-vision*.nl

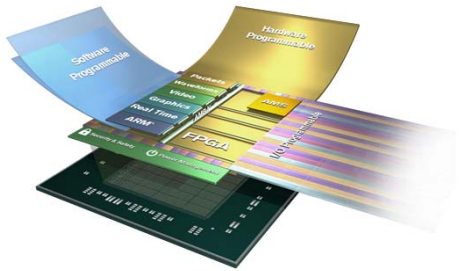
Training Program

Essentials of FPGA Design	*1 day*
Designing for Performance	*2 days*
Advanced FPGA Implementation	*2 days*
Design Techniques for Lower Cost	*1 day*
Designing with Spartan-6 and Virtex-6 Family	*3 days*
Essential Design with the PlanAhead Analysis Tool	*1 day*
Advanced Design with the PlanAhead Analysis Tool	*2 days*
Xilinx Partial Reconfiguration Tools and Techniques	*2 days*
Designing with the 7 Series Families	*2 days*



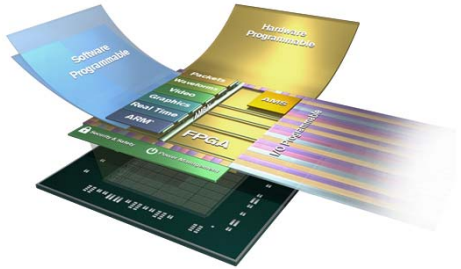
Training Program

- Designing FPGAs Using the Vivado Design Suite 1 *2 days*
- Designing FPGAs Using the Vivado Design Suite 2 *2 days*
- Designing FPGAs Using the Vivado Design Suite 3 *2 days*
- Designing FPGAs Using the Vivado Design Suite 4 *2 days*
- Designing with the UltraScale and UltraScale+ Architecture *2 days*
- Vivado Design Suite for ISE Software Project Navigator User *1 day*
- Vivado Design Suite Advanced XDC and Static Timing Analysis for ISE Software User *2 days*



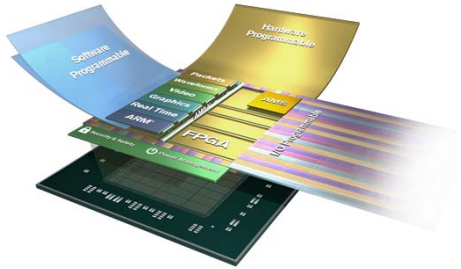
Training Program

- Designing with Multi Gigabit Serial IO *3 days*
- High Level Synthesis with Vivado *2 days*
- C-Based HLS Coding for Hardware Designers *1 day*
- C-Based HLS Coding for Software Designers *1 day*
- DSP Design Using System Generator *2 days*
- Essential DSP Implementation Techniques for Xilinx FPGAs *2 days*



Training Program

- Embedded Systems Design *2 days*
- Embedded Systems Software Design *2 days*
- Advanced Features and Techniques of SDK *2 days*
- Advanced Features and Techniques of EDK *2 days*
- Zynq All Programmable SoC Systems Architecture *2 days*
- Zynq UltraScale+ MPSoC for the System Architect *2 days*
- Introduction to the SDSoC Development Environment *1 day*
- Advanced SDSoC Development Environment & Methodology *2 days*



Training Program

VHDL for Designers	*3 days*
Advanced VHDL	*2 days*
Comprehensive VHDL	*5 days*
Expert VHDL Verification	*3 days*
Expert VHDL Design	*2 days*
Expert VHDL	*5 days*
Essential Digital Design Techniques	*2 days*