SDSoC Technical Seminars 2016

Feb 2016

Agenda

- Zynq SoC and MPSoC Architecture
- SDSoC Overview
- Real-life Success
- C/C++ to Optimized System
- Targeting Your Own Platform
- Next Steps

Agenda

- **Zynq SoC and MPSoC Architecture**
- SDSoC Overview
- Real-life Success
- C/C++ to Optimized System
- Targeting Your Own Platform
- Next Steps

Zynq-7000: The First All Programmable SoC

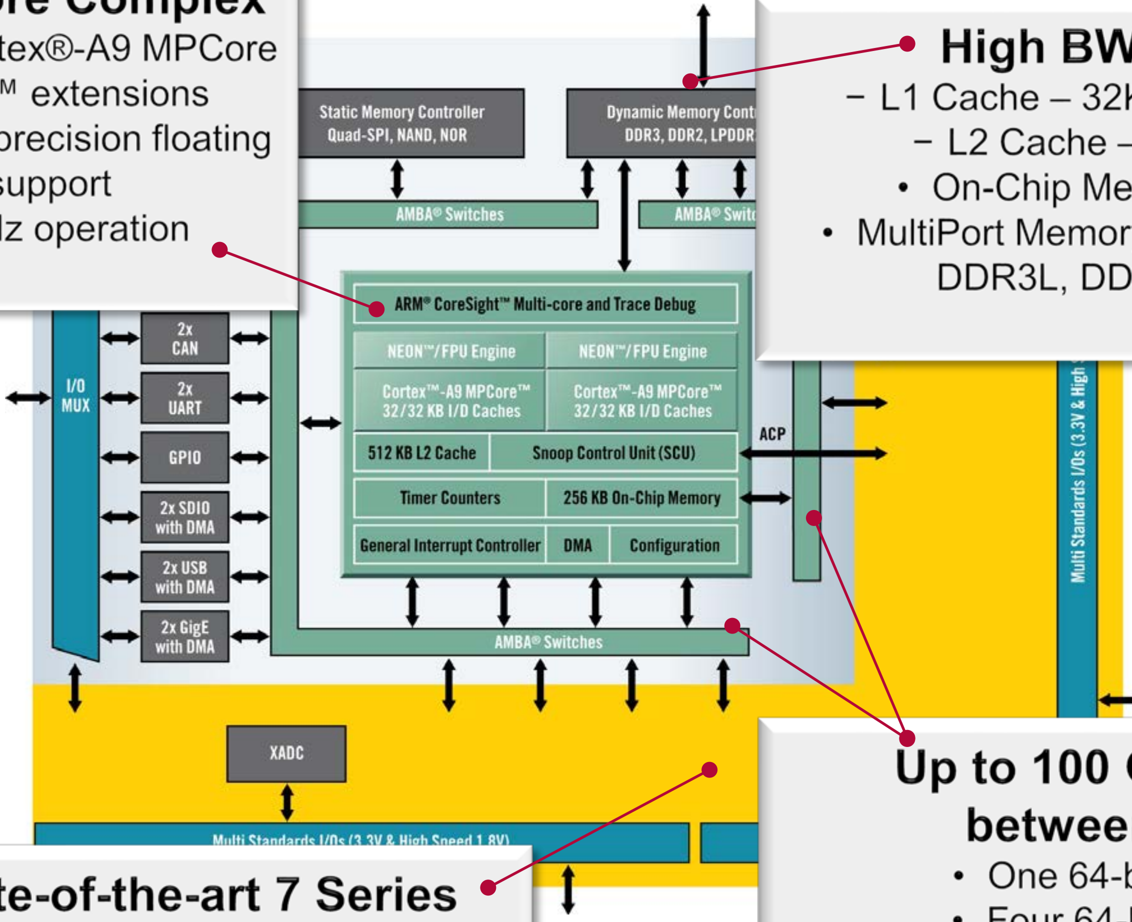
Security Integration Power Performance BOM Cost

Zynq®-7000 SoC

- **Innovative ARM + FPGA architecture on a single die**
- **Reduce BOM cost by replacing multiple chips with a single Zynq**
- **Security through single chip solution and secure boot**
- **Remove off-chip communication bottleneck**
- **Architecture optimize for power**

Delivering Future Generations of Smarter and Optimized SoCs

Zynq®-7000 Architeture



Processor Core Complex

- Dual ARM® Cortex®-A9 MPCore with NEON™ extensions
- Single / double precision floating point support
- Up to 1 GHz operation

High BW Memory

- L1 Cache – 32KB/32KB (per Core)
- L2 Cache – 512KB Unified
- On-Chip Memory of 256KB
- MultiPort Memory Controller (DDR3, DDR3L, DDR2, LPDDR2)

Up to 100 Gb/s of BW between PS-PL

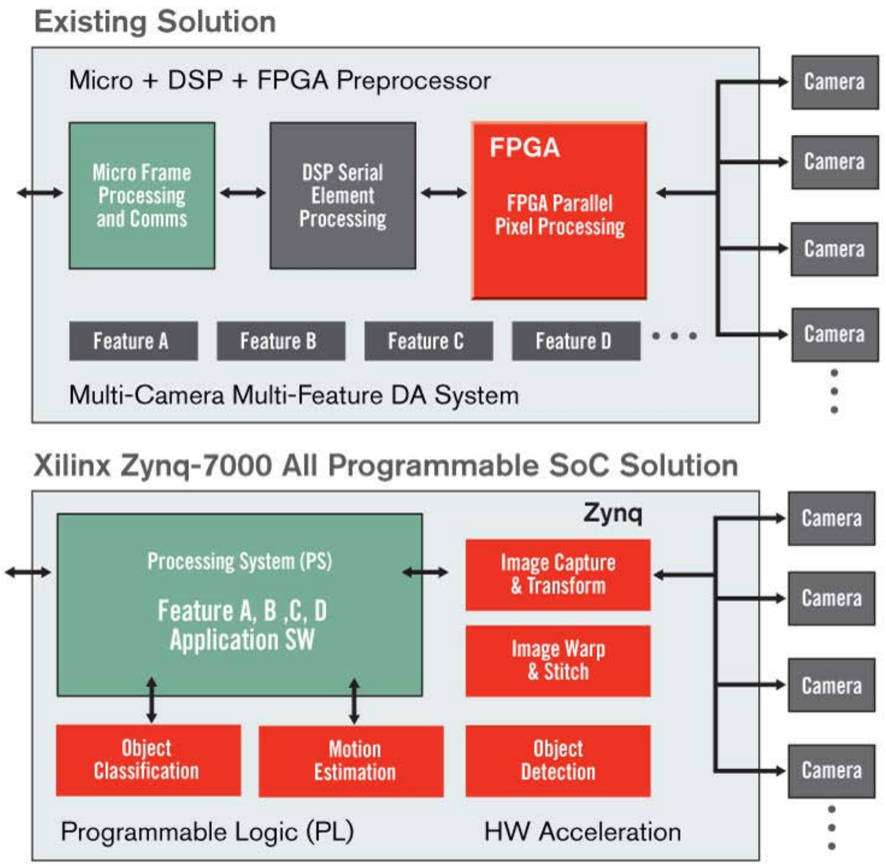
- One 64-bit ACP Port
- Four 64-bit HP Ports
- Four 32-bit GP Ports

State-of-the-art 7 Series Programmable Logic

- 28K-440K logic cells
- 430K-6.6M equivalent ASIC gates
- Hard PCIe Core Gen2x8

Success Story 1: Automotive ADAS Platform

- **Integrated into a single monolithic device**
  – Sensing domain
  – Environmental characterization
  – Decision-making features

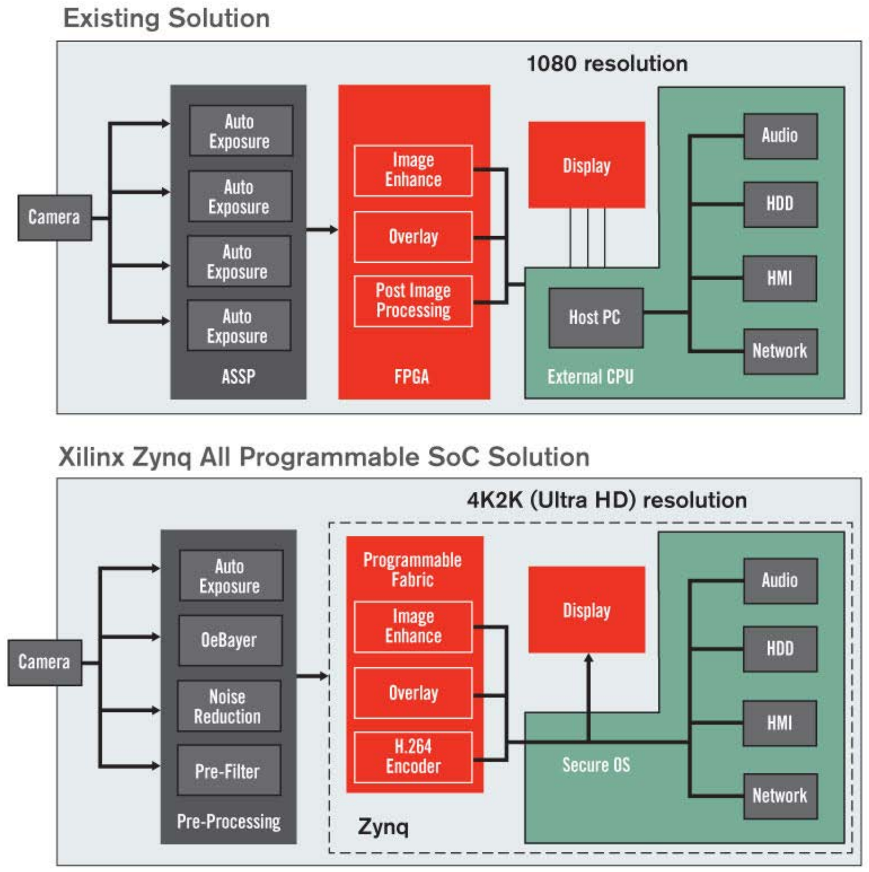

Existing Solution
Micro + DSP + FPGA Preprocessor
Micro Frame Processing and Comms
DSP Serial Element Processing
FPGA
FPGA Parallel Pixel Processing
Feature A
Feature B
Feature C
Feature D
Multi-Camera Multi-Feature DA System
Camera
Camera
Camera
Camera
Xilinx Zynq-7000 All Programmable SoC Solution
Zynq
Processing System (PS)
Feature A, B ,C, D Application SW
Image Capture & Transform
Image Warp & Stitch
Object Classification
Motion Estimation
Object Detection
Programmable Logic (PL)
HW Acceleration
Camera
Camera
Camera
Camera

Solution Benefits
System Integration **3 chips ⇨ 1 chip**
System Performance **+2x**
BOM Cost **-25%**
Total Power **-50%**

XILINX ALL PROGRAMMABLE.

Success Story 2: 1080p60 HD Medical Endoscope

- **Integrated into a single monolithic device**
  - Camera unit control
  - 1080p60 image processing
  - Hardware acceleration of complex video analytics



Solution Benefits
System Integration **2 chips ⇨ 1 chip**
System Performance **+7x Bandwidth**
BOM Cost **-10%**
Total Power **-35%**

XILINX ALL PROGRAMMABLE.

Industry Trends and Customer Challenges

Packet Processing & Transport

- > 400G OTN
- Video Over IP
- Software Defined Networks
- Network Function Virtualization

Wireless

- LTE Advanced
- Cloud-RAN
- Early 5G
- Heterogeneous Wireless Networks

Video and Vision

- 8K/4K Resolution
- Immersive Display
- Augmented Reality
- Video Analytics

Cloud and Data Center

- Acceleration
- Big Data
- Software Defined Data Center
- Public and Private Cloud

Industrial IoT

- Machine to Machine
- Sensory Fusion
- Industry 4.0
- Cyber-Physical
- Embedded Vision

1 Performance & Power Scalability

2 System Integration & Intelligence

3 Security, Safety & Reliability

XILINX ALL PROGRAMMABLE.

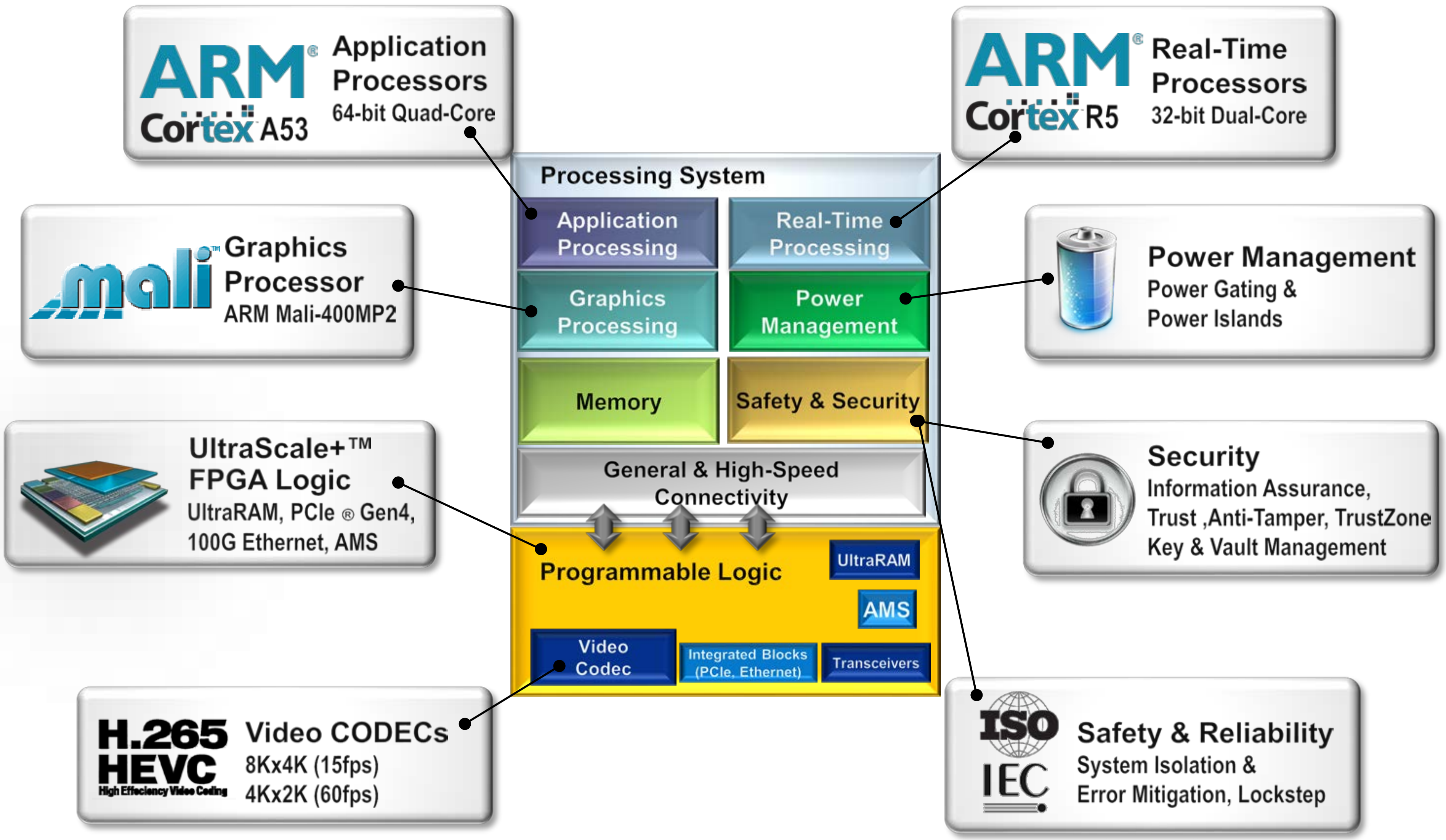
Introducing UltraScale+™ MPSoC

Zynq®
All
Programmable
MPSoC

ZYNQ
UltraSCALE+

- ✓ 2-5X System Performance/Watt
- ✓ Most System Integration & Intelligence
- ✓ Highest Levels of Security and Safety

Zynq® UltraScale+ MPSoC System Features



Zynq® UltraScale+™ MPSoC Applications

1 Terabyte OTN Switching

800G MAC-Interlaken Bridge

800G Data Center Interconnect

Mobile Backhaul
1 GHz eBand Modem & Packet Processing

Mobile Backhaul
112 MHz PtP MWR Modem & Packet Processing

Test & Measurement Instrumentation

24-Channel Radar
(Beamformer + Pulse Compressor + Doppler Filter)

8x8 100 MHz LTE Remote Radio Head

Dual-Channel Battery-Powered
Public Safety & Military Mobile Radios

Camera-Based Automotive
Driver's Assist Systems (ADAS)

4K Broadcast Cameras

Solid State Drives (SSDs) for Data Center

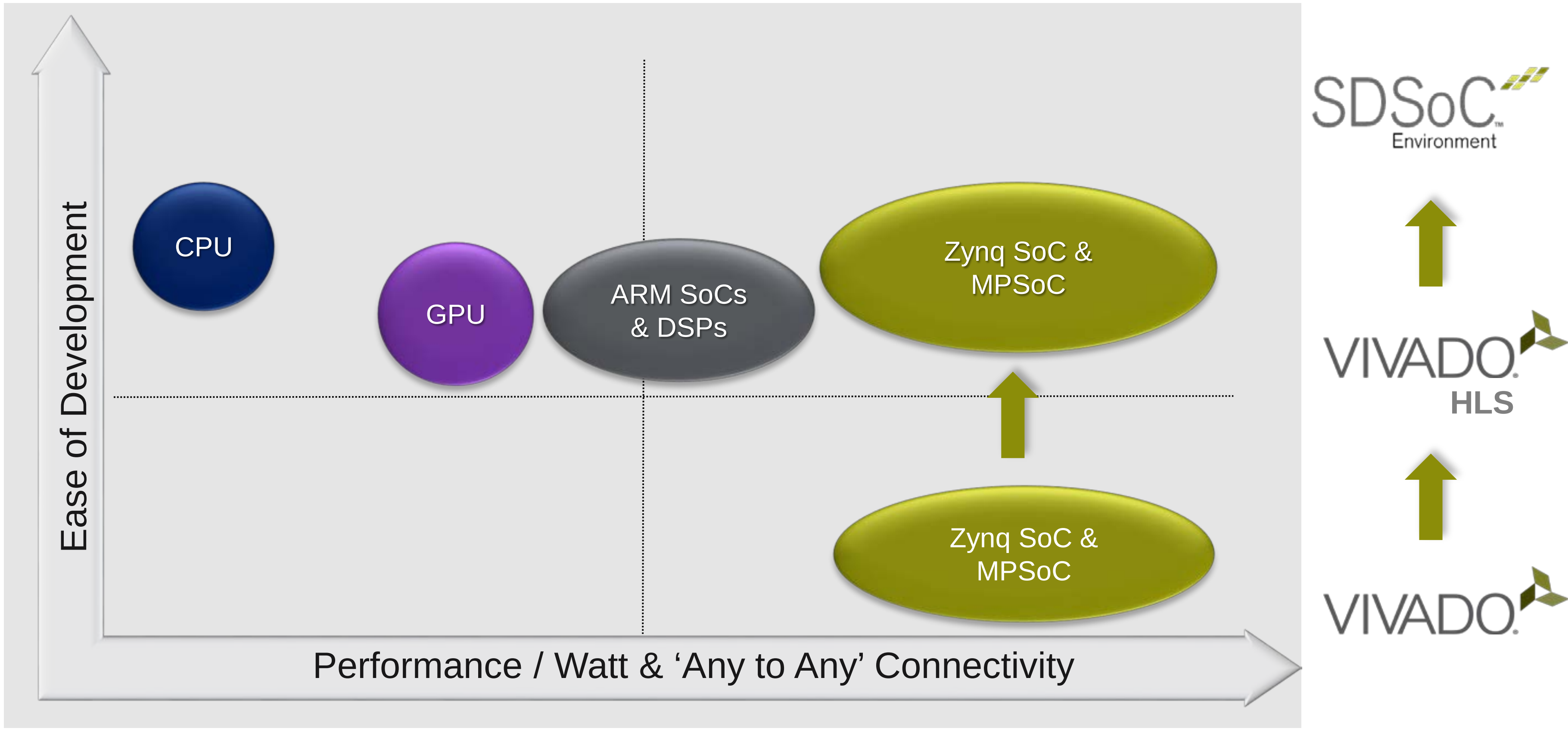
Video Conferencing

High-Performance Scalable
Programmable Logic Controllers (PLCs)

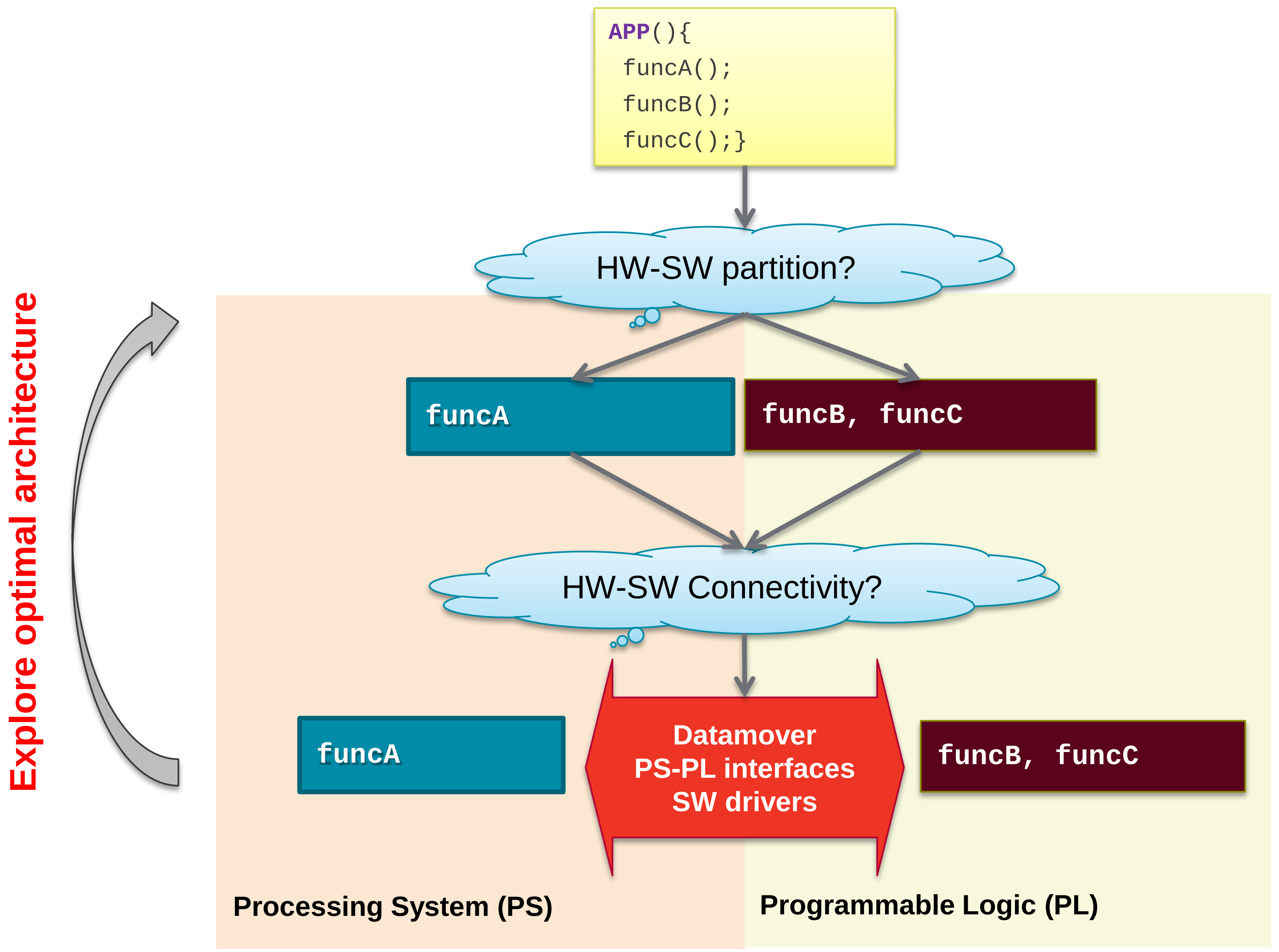
Agenda

- Zynq SoC and MPSoC Architecture
- **SDSoC Overview**
- Real-life Success
- C/C++ to Optimized System
- Targeting Your Own Platform
- Next Steps

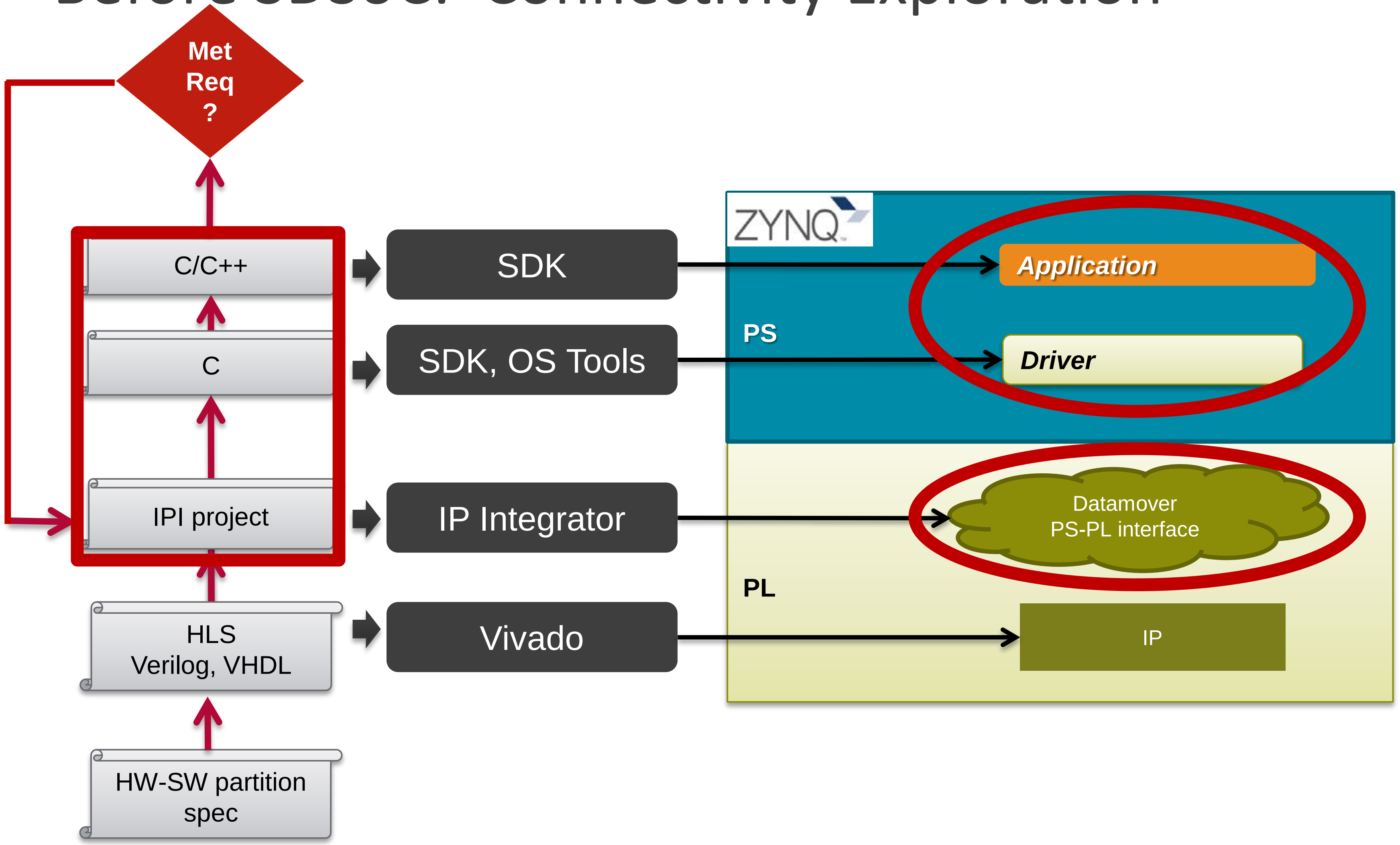
Scaling Productivity with Technology Advancement



Typical Zynq Development Flow

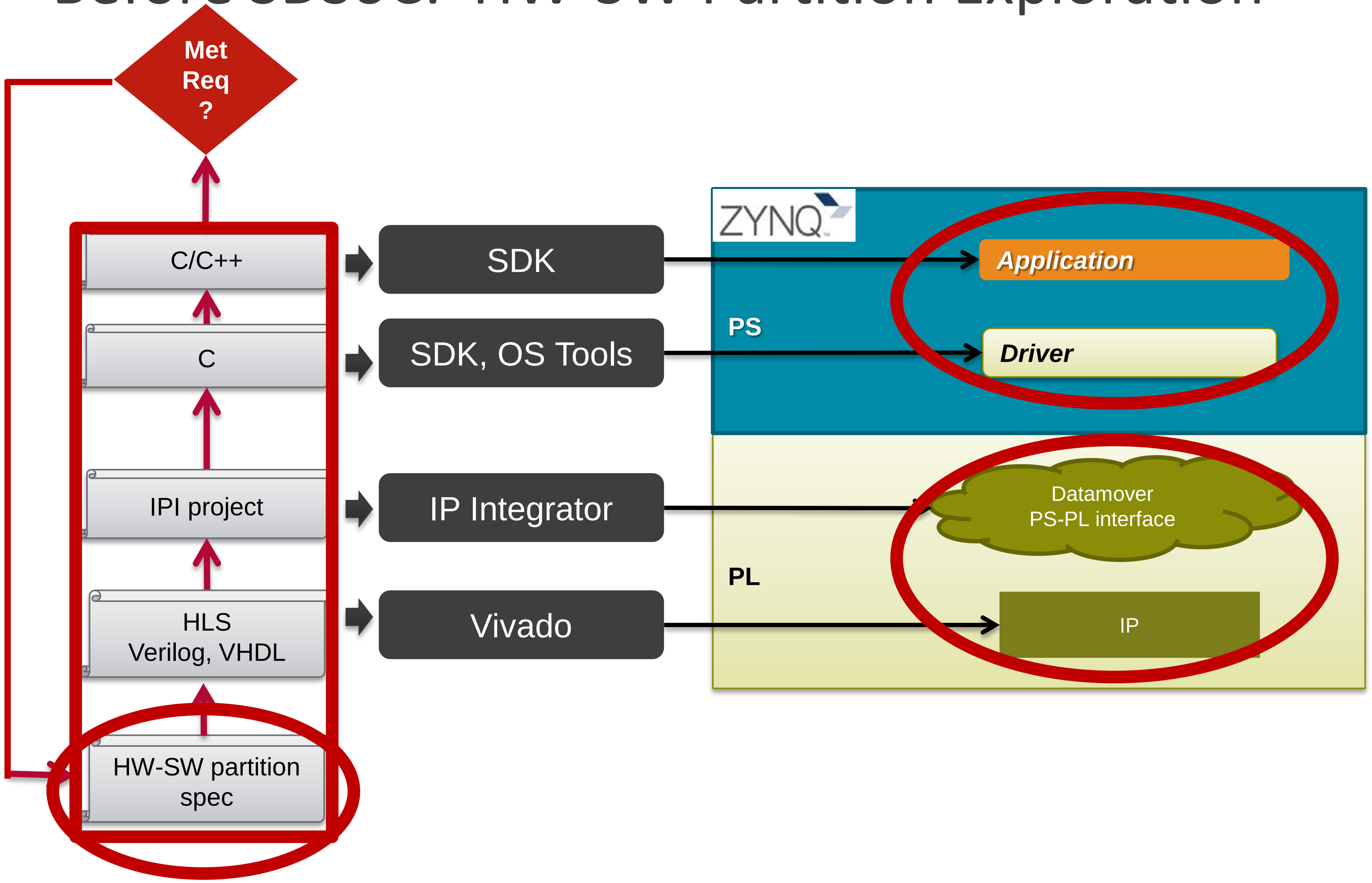


Before SDSoC: Connectivity Exploration



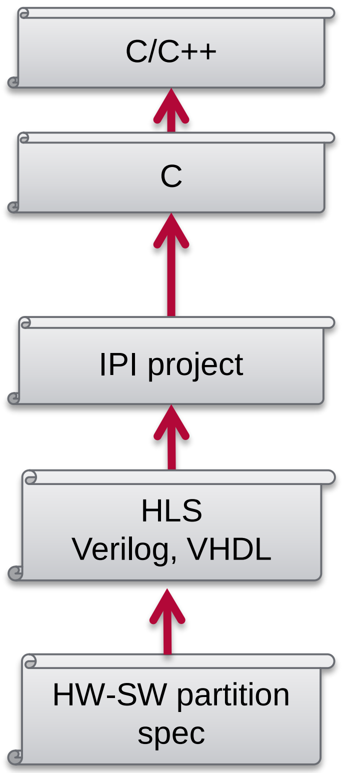
Need to modify multiple levels of design entry

Before SDSoC: HW-SW Partition Exploration



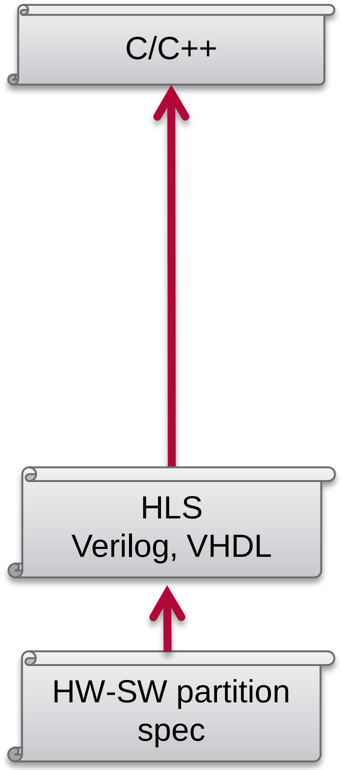
Involving multiple discipline to explore architecture

After SDSoC:

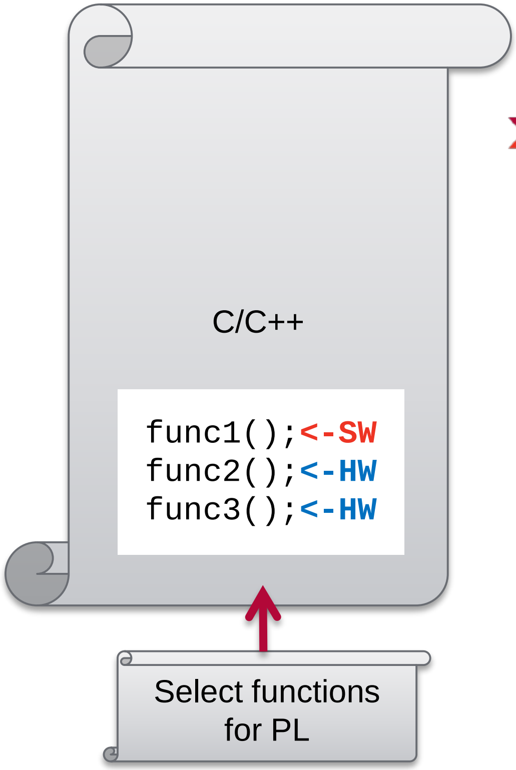


After SDSoC:

- Auto-generate SW drivers and HW connectivity



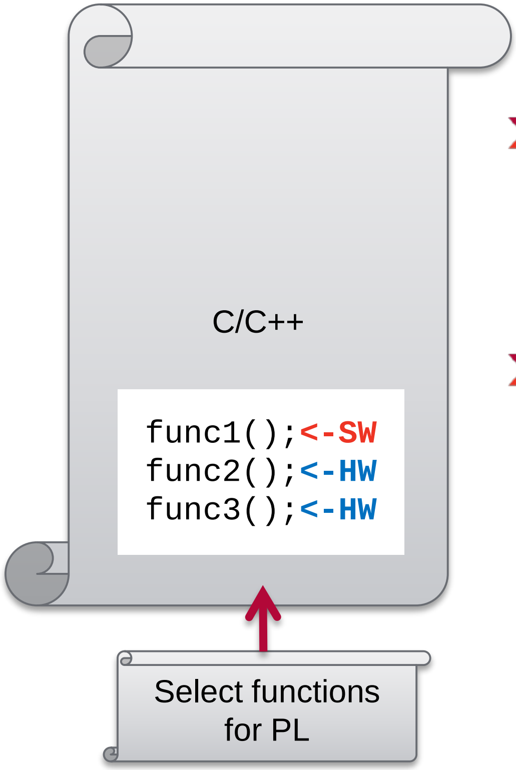
After SDSoC:



```
func1();<-SW
func2();<-HW
func3();<-HW
```

- **Auto-generate SW drivers and HW connectivity**
- **Select hardware accelerator functions in GUI or via command line and C-callable libraries**

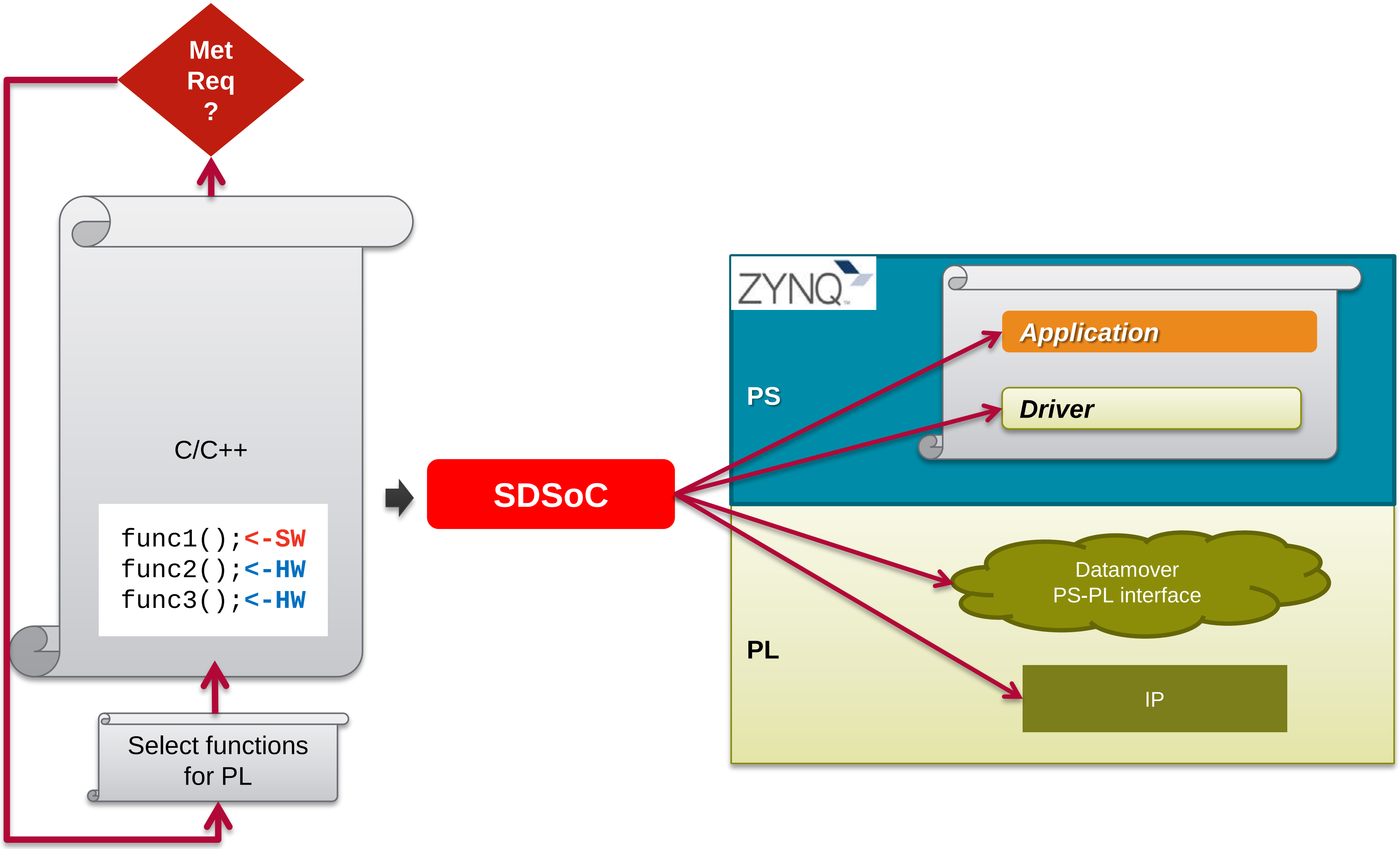
After SDSoC:



```
func1();<-SW
func2();<-HW
func3();<-HW
```

- **Auto-generate SW drivers and HW connectivity**
- **Select hardware accelerator functions in GUI or via command line and C-callable libraries**
- **HW function calls in application C/C++ code defines the hardware / software partition**

After SDSoC: Automatic System Generation

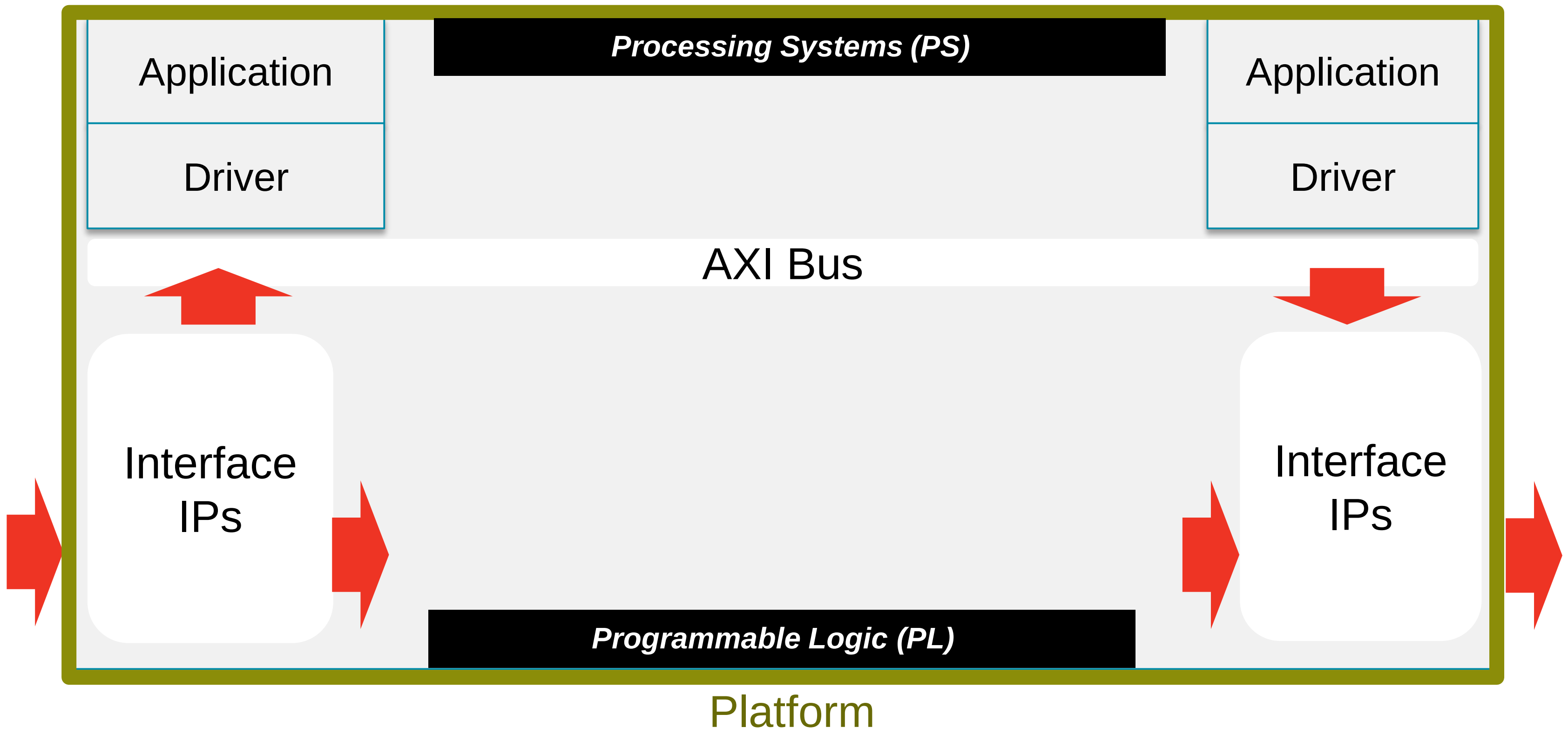


C/C++ to running system in hours, days

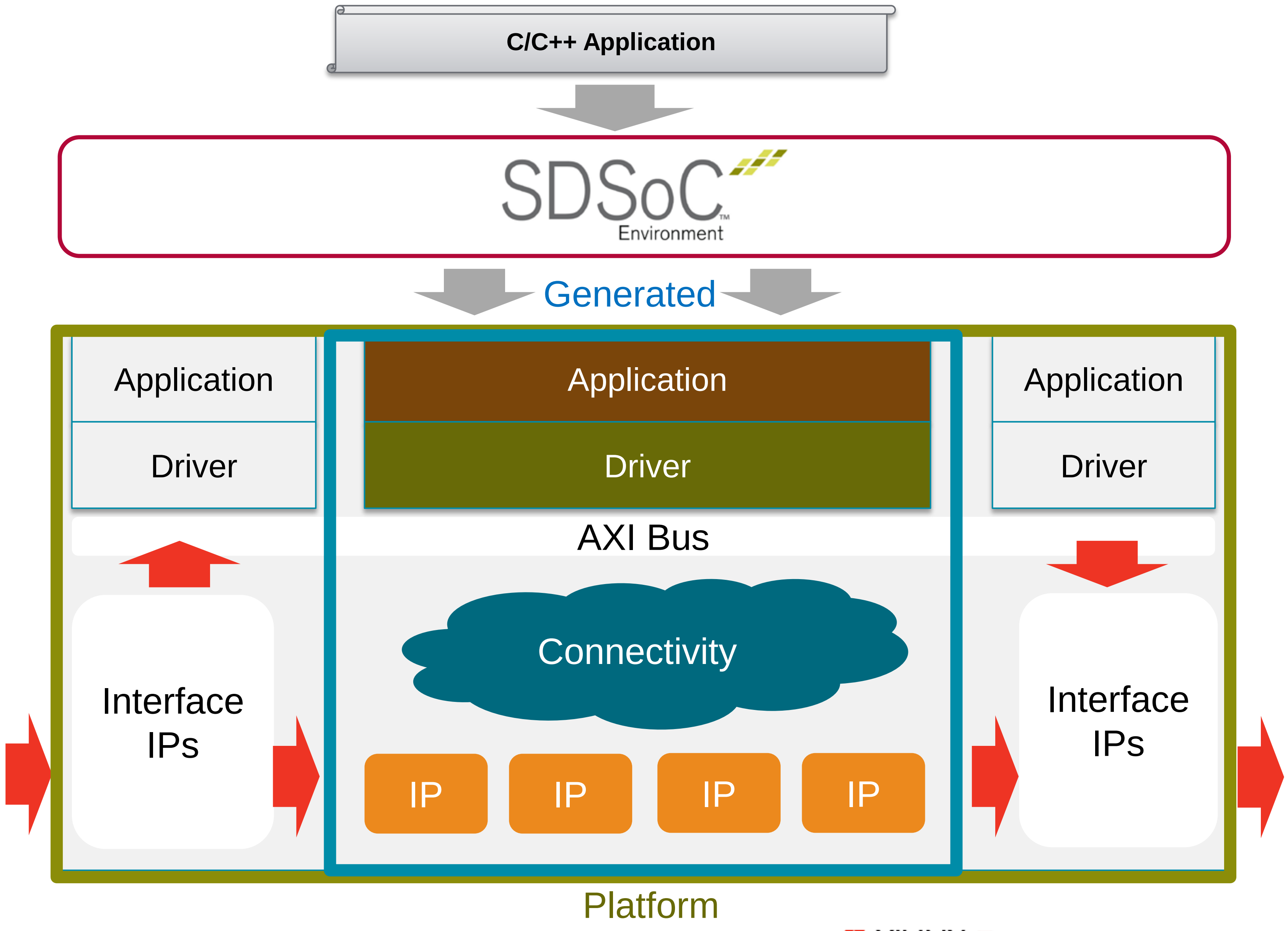
XILINX ALL PROGRAMMABLE.

Base Platform

- **Platform: base HW system, OS, bootloaders, file system, libraries**
  - Processing system, memory interfaces, custom I/O, other subsystems, …



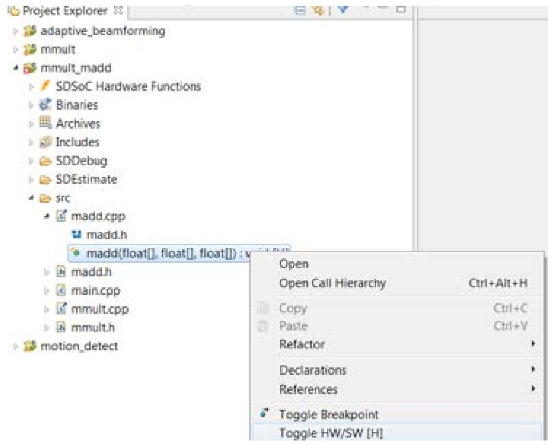
Complete End-to-End Flow



A Complete Software Development Environment

C/C++ Application

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



```
Project Explorer
 adaptive_beamforming
 mmult
 mmult_madd
   SDSoC Hardware Functions
   Binaries
   Archives
   Includes
   SDDebug
   SDEstimate
   src
     madd.cpp
       madd.h
       madd(float[], float[], float[]) : v
     madd.h
     main.cpp
     mmult.cpp
     mmult.h
 motion_detect
```

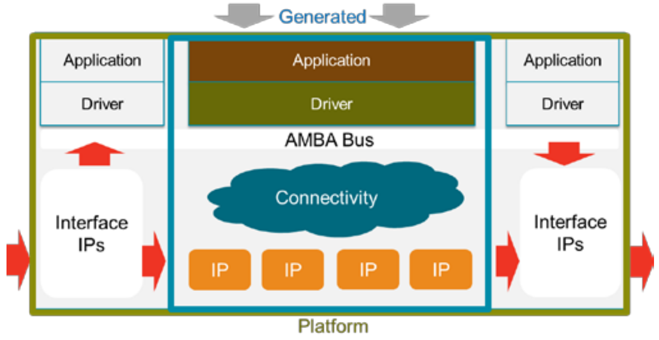
```
Open
Open Call Hierarchy        Ctrl+Alt+H
Copy                       Ctrl+C
Paste                      Ctrl+V
Refactor
Declarations
References
Toggle Breakpoint
Toggle HW/SW [H]
```

TCF Profiler

Profiler running, [illegible] samples

%	% Inclusive	Function
[illegible]	[illegible]	main
[illegible]	[illegible]	sobel_operator
14.8	14.8	rgb_pad2ycbcr
10.7	10.7	getval
5.10	5.10	ycbcr2rgb_pad
[illegible]	[illegible]	sobel_filter

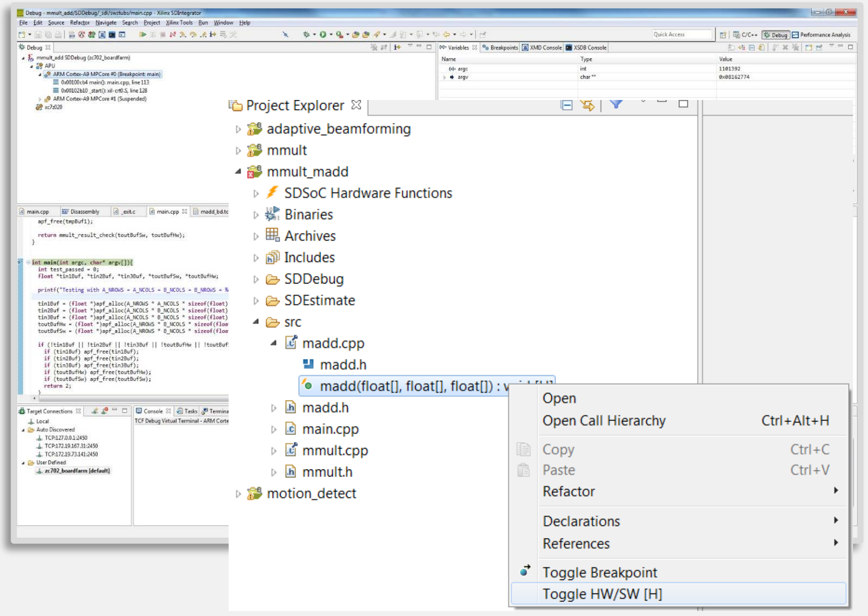
Proven Xilinx tools in the backend



XILINX ALL PROGRAMMABLE.

SDSoC's Software Programming Experience

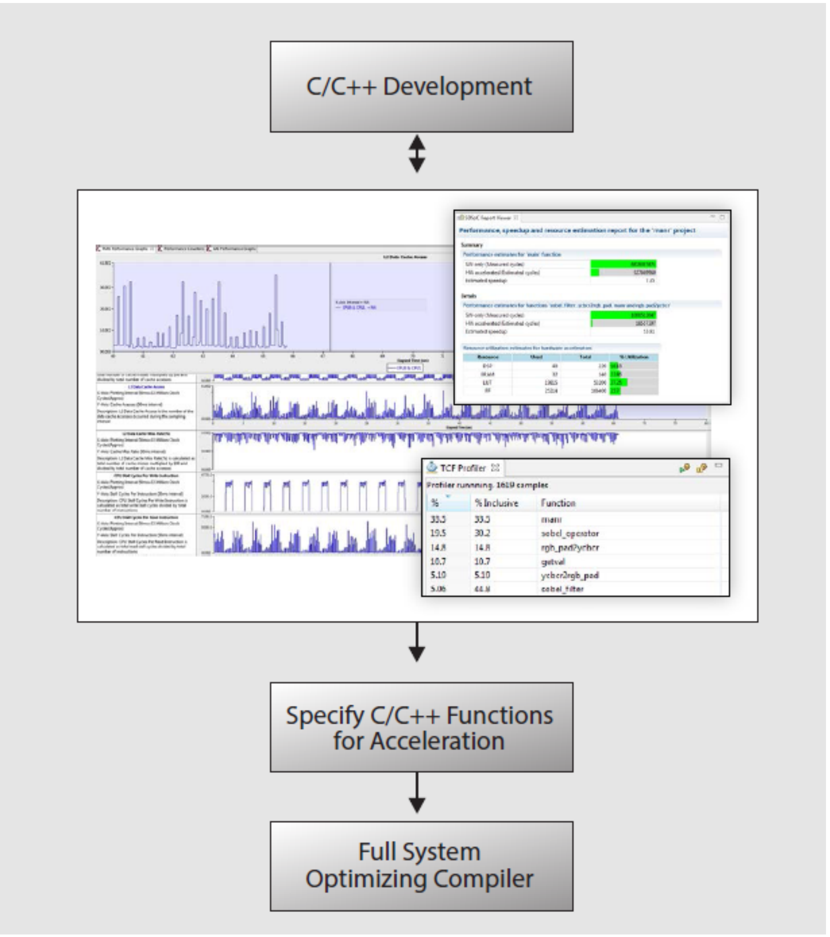
C/C++ Development



- User selects C/C++ functions to accelerate in programmable logic (PL)
- C/C++ system compiler and linker (CLI)
- Easy to use Eclipse IDE
- Optimized HLS libraries
- Support for target Linux, bare metal, FreeRTOS

XILINX ALL PROGRAMMABLE.

SDSoC's System Level Profiling

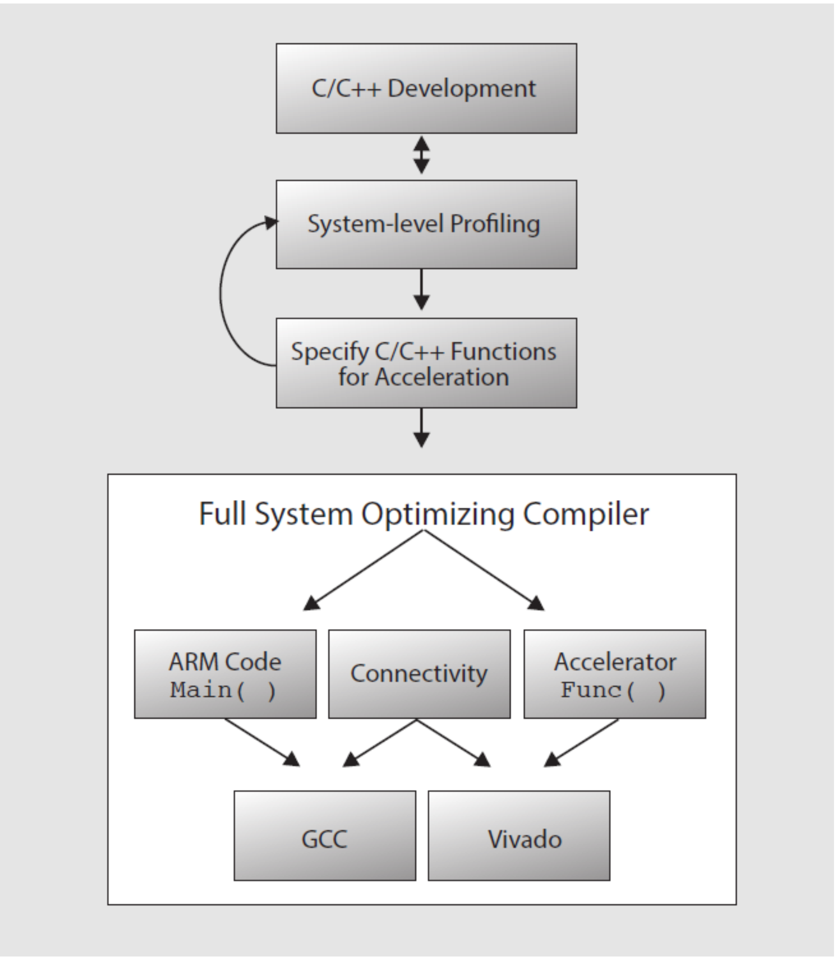

C/C++ Development
Specify C/C++ Functions for Acceleration
Full System Optimizing Compiler
TCF Profiler
Performance, speedup and resource estimation report for the 'main' project

Profiler running. 1619 samples

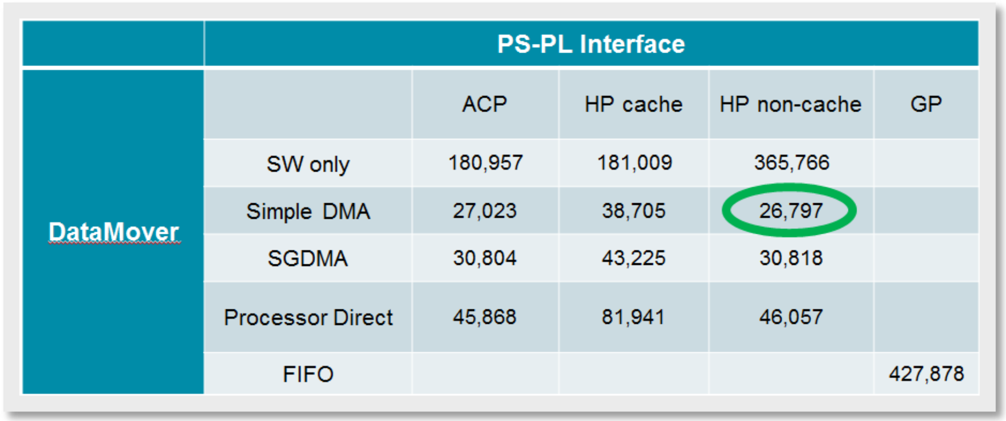
%	% Inclusive	Function
33.3	33.3	main
19.5	30.2	sobel_operator
14.8	14.8	rgb_pad2ycbcr
10.7	10.7	getval
5.10	5.10	ycbcr2rgb_pad
5.06	44.8	sobel_filter

- **Rapid system estimation**
  - Pre-RTL synthesis, place & route
  - Fast design feedback on accelerator network
- **Automated performance measurement**
  - Runtime measurement of cache, memory, and bus utilization
  - Combined HW-SW event trace of accelerator network

Full System Optimizing Compiler



- **Full system from C/C++**
  - Optimizing programmable logic fabric cross-compiler (HLS)
  - Automatic data motion network inference
  - Application-specific system optimized for latency and throughput/
  - User override via C/C++ pragmas

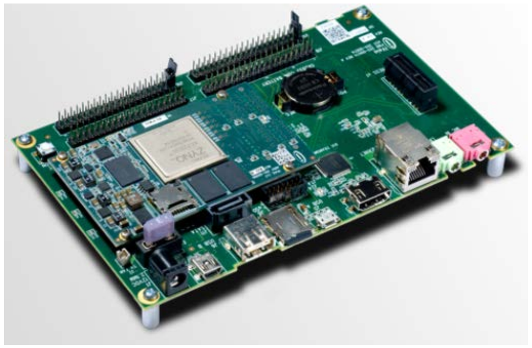


		PS-PL Interface			
		ACP	HP cache	HP non-cache	GP
DataMover	SW only	180,957	181,009	365,766	
	Simple DMA	27,023	38,705	26,797	
	SGDMA	30,804	43,225	30,818	
	Processor Direct	45,868	81,941	46,057	
	FIFO				427,878

Available SDSoC Platforms

Video Platforms (Externally Provided)

Board Name & Description	**I/O enabled**
ZC702 + HDMI IO FMC	HDMI in, HDMI out, PS DDR
ZC706 + HDMI IO FMC	HDMI in, HDMI out, PS DDR
Smart Vision Development Kit (SVDK)	Camera in, GigEV out, PS DDR
Atlas-I-Z7e + Captiva Carrier Card	GigEV in, HDMI out, PS DDR
MIAMI	PS DDR
Zing2 + HDMI IO FMC	HDMI IN, HDMI OUT, GPIO,PS,DDR3
Snowleo SVC	CMOS IN,HDMI OUT,GPIO,PS,DDR3
EMC2-Z7015	PS DDR
BORA	LVDS Video Out, PS DDR
BORA Xpress	LVDS Video Out, PS DDR
ZYBO	HDMI in, VGA out, buttons, switches, LEDs



Radio Platforms (Externally Provided)

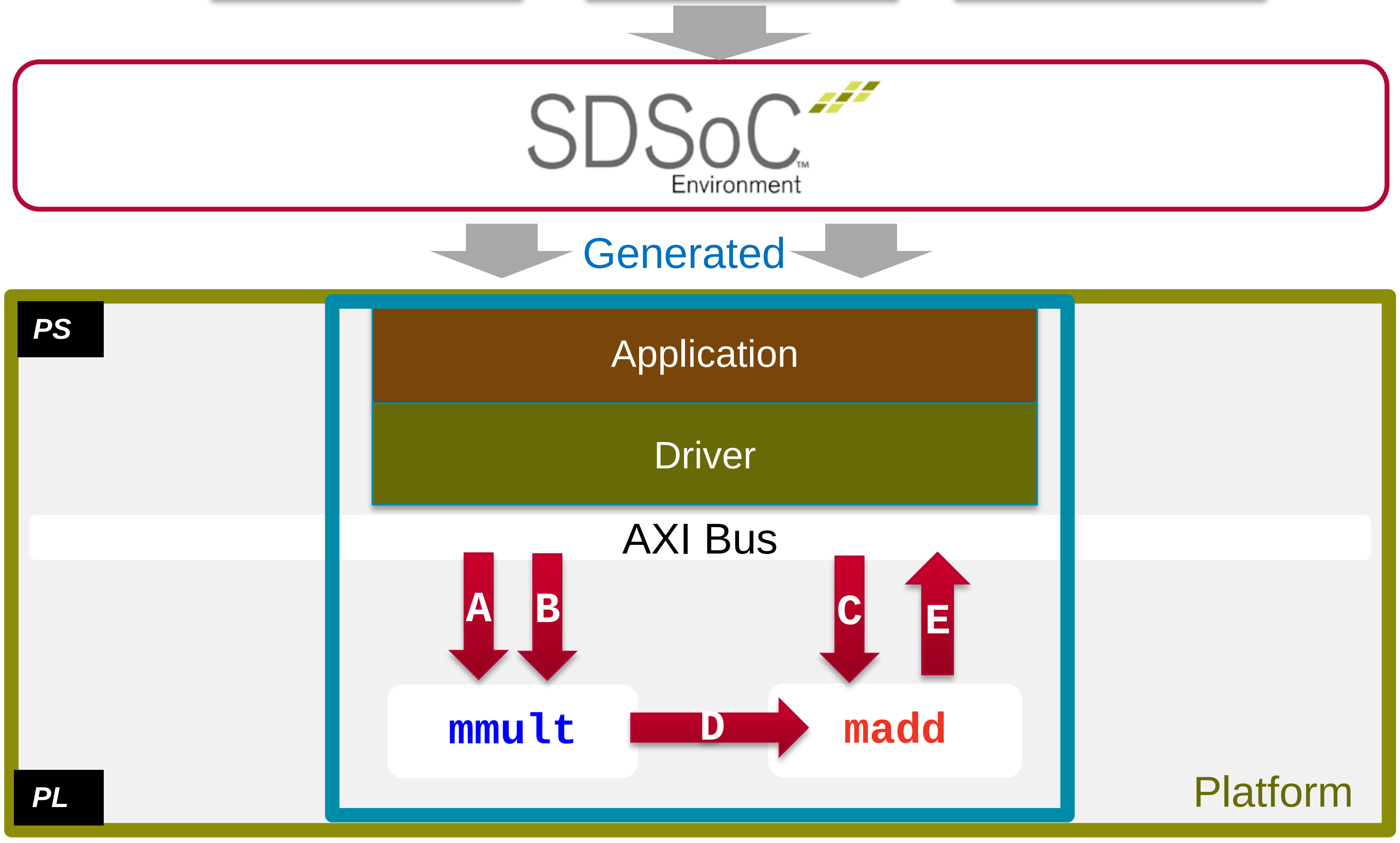
Board Name & Description	**I/O enabled**
Atlas-II-Z7x + Mosaic carrier card	ADC, DAC, PS DDR
ZC706 + AD9361 SDR Systems Development Kit	ADC, DAC, PS DDR

Example 1: Matrix Multiply + Add

```
madd(inA,inB,out){
    HLS C/C++
}
```

```
main(){
 malloc(A,B,C);
 mmult(A,B,D);
 madd(C,D,E);
 printf(E);
}
```

```
mmult(inA,inB,out){
    HLS C/C++
}
```

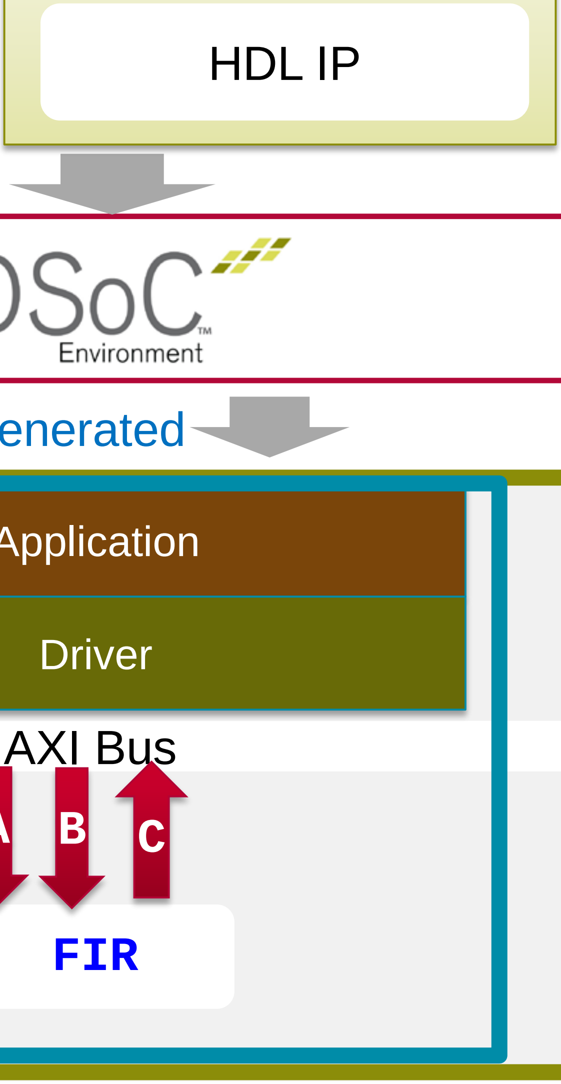


datamovers

Example 2: FIR Filter using C-callable HDL IP

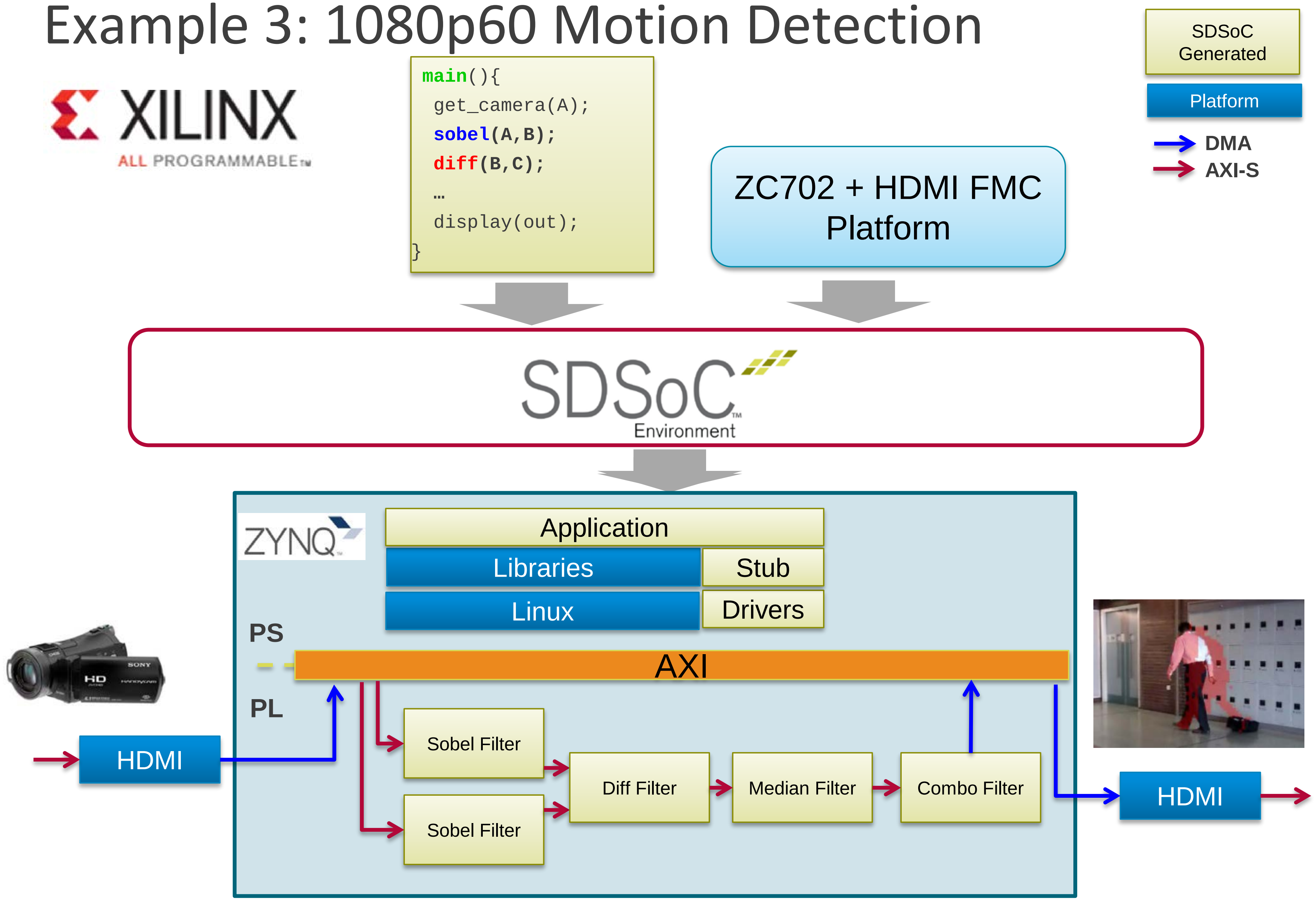
```
main(){
 malloc(A,B,C);
 fir_config(A);
 fir_run(B,C);
 printf(C);
}
```

```
fir_config(float *coef);
fir_run(float *in, float *out);
```



datamovers

Example 3: 1080p60 Motion Detection

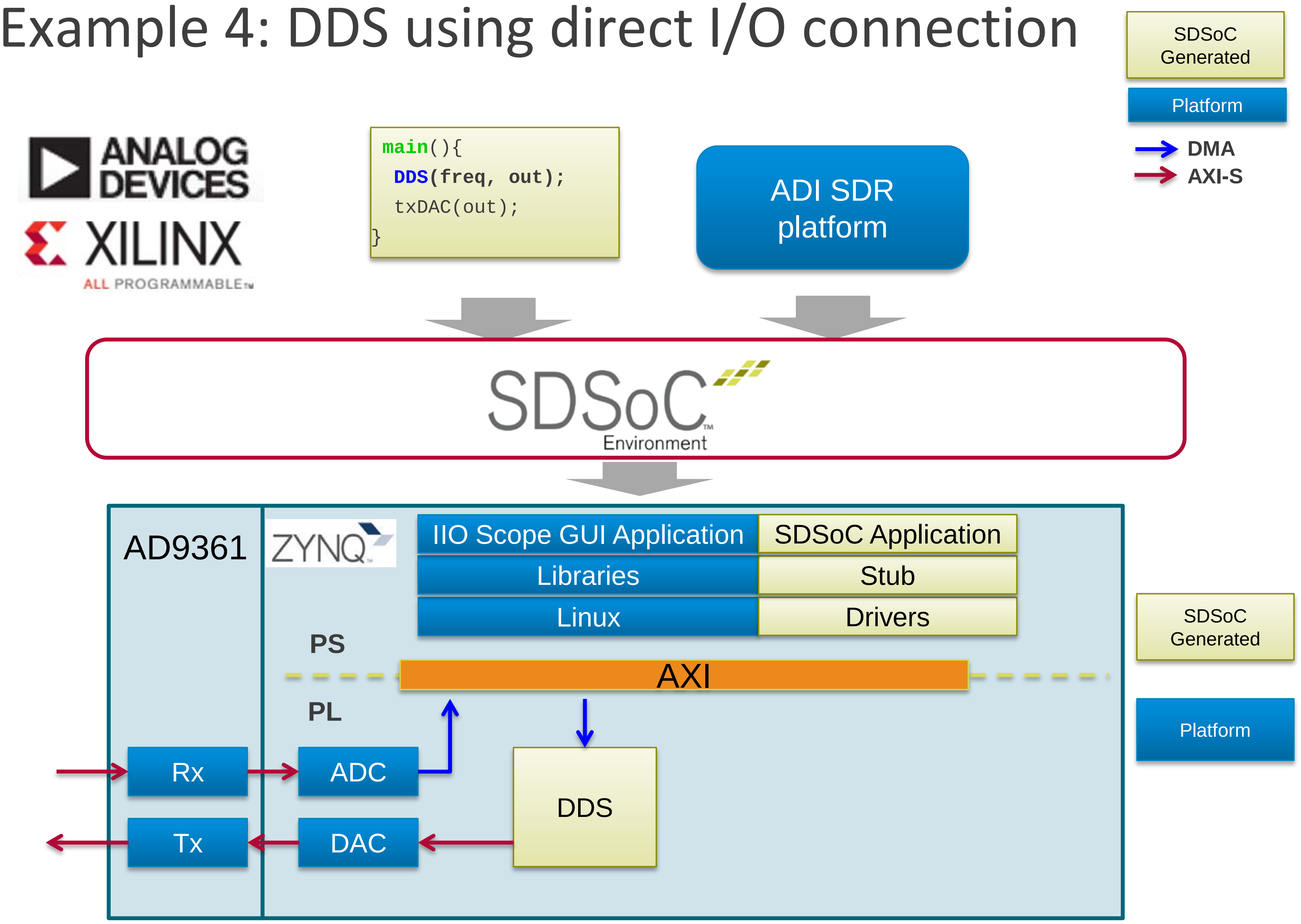


```
main(){
  get_camera(A);
  sobel(A,B);
  diff(B,C);
  …
  display(out);
}
```

Image processing on the video I/Os via DDR3 memory

Example 4: DDS using direct I/O connection

```
main(){
  DDS(freq, out);
  txDAC(out);
}
```



Direct I/O connection to the platform DAC

Agenda

- Zynq SoC and MPSoC Architecture
- SDSoC Overview
- **Real-life Success**
- C/C++ to Optimized System
- Targeting Your Own Platform
- Next Steps

Object Recognition

SDSoC Generated

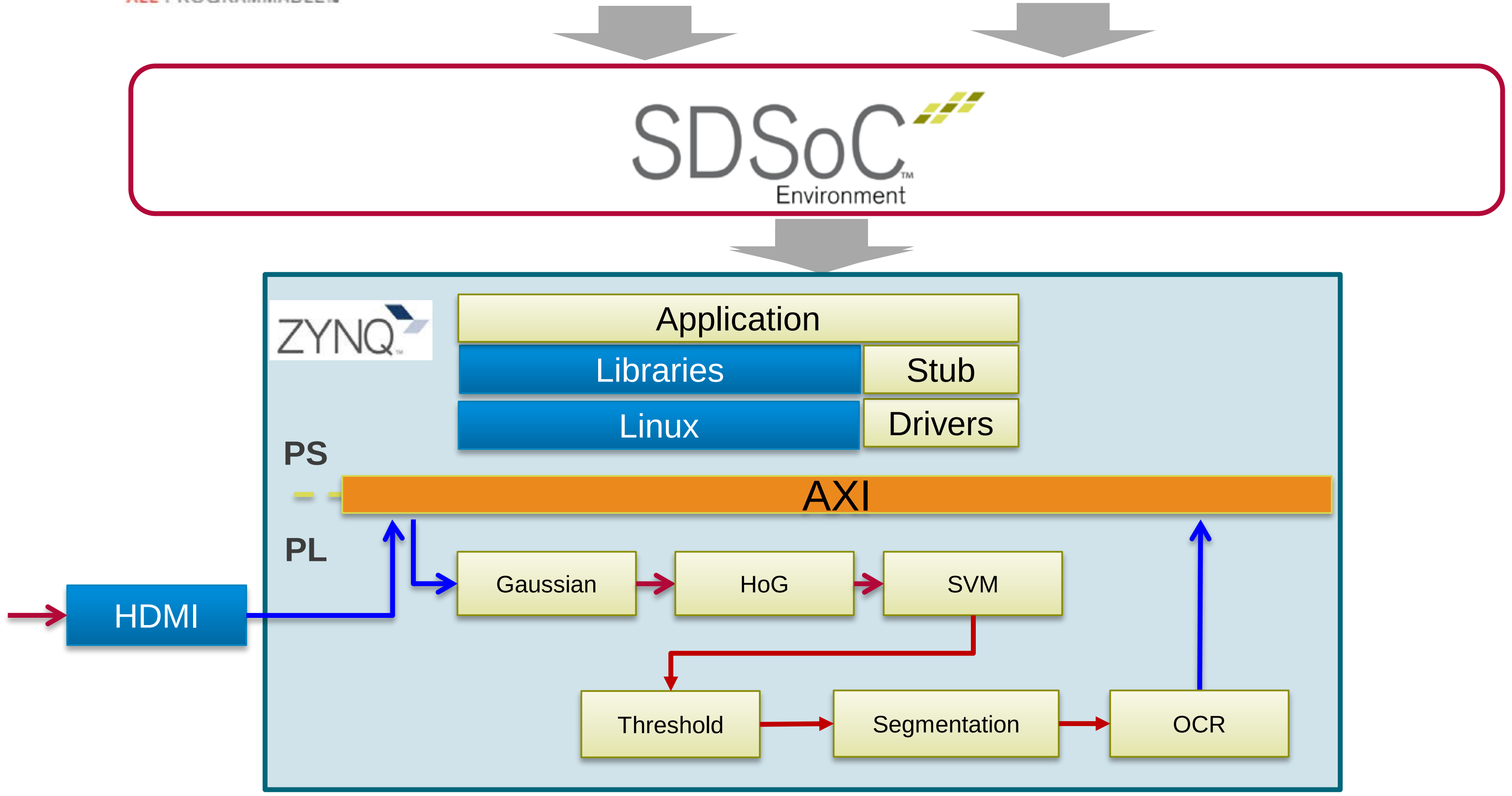
Platform

DMA

AXI-S

```
main(){
  get_camera(A);
  auANPRTop (A,B);
  display(out);
}
```

ZC706 + HDMI FMC Platform



Uses HW Optimized OpenCV Libraries

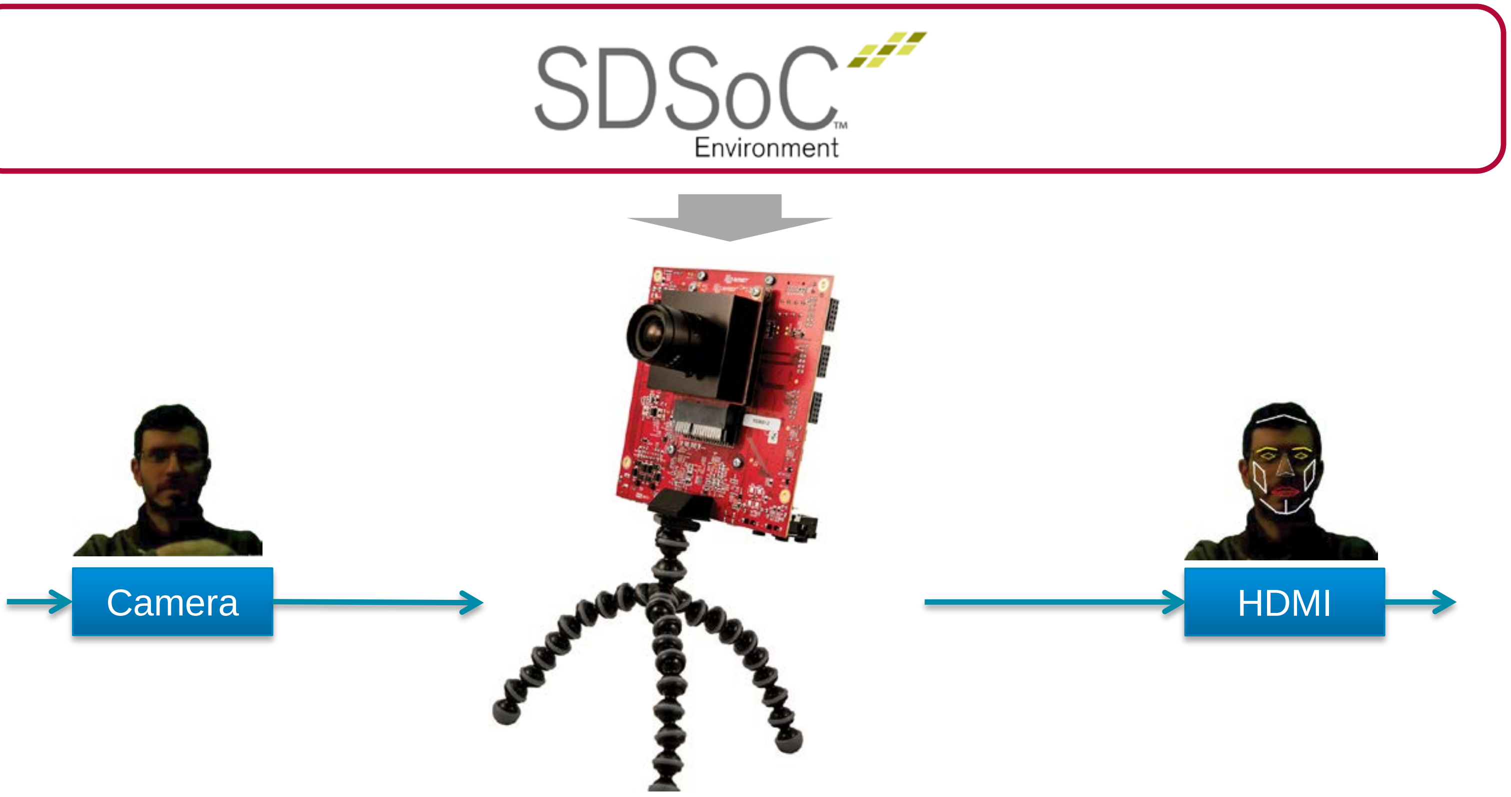
Hardware Optimized OpenCV Libraries

Auviz Systems

Computations	Input processing	Filter	Other	Other
Absolute difference	Channel combine	Box	Canny edge detection	Histogram of Gradients (HoG)
Accumulate	Channel extract	Gaussian	Scale/Resize	ORB
Accumulate squared	Color convert	Median	Warp Affine	SVM (binary)
Accumulate weighted	Convert bit depth	Sobel	Warp Perspective	OTSU Thresholding
Arithmetic addition	Table lookup	Custom convolution	Image pyramid	Mean Shift Tracking (MST)
Arithmetic subtraction	Histogram		Fast corner	LK Optical Flow
Bitwise: AND, OR, XOR, NOT	Gradient Phase	Dilate	Harris corner	
Pixel-wise multiplication	Min/Max Location	Erode	Remap	
Integral image	Mean & Standard Deviation	Bilateral	Equalize Histogram	
Gradient Magnitude	Thresholding			

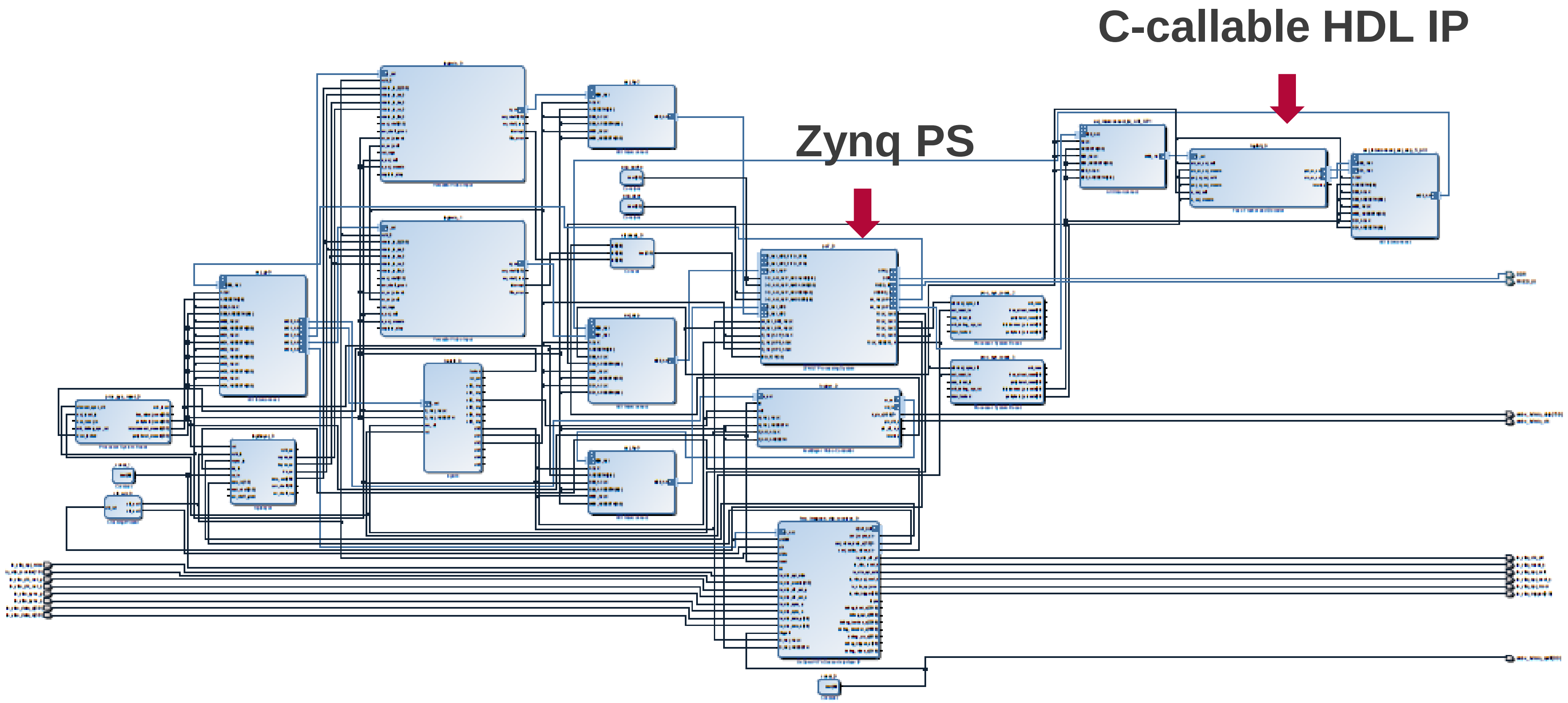
Face Detection and Tracking

```
main(){
 getCamera(A);
 while (FaceFeatures){
   face_detect(A,B);
 }
 DrawFacial(A,C);
 Display;}
```



Uses Optimized HDL IP as a C function in SDSoC

Automatically Generated Vivado Design from C/C++ Application

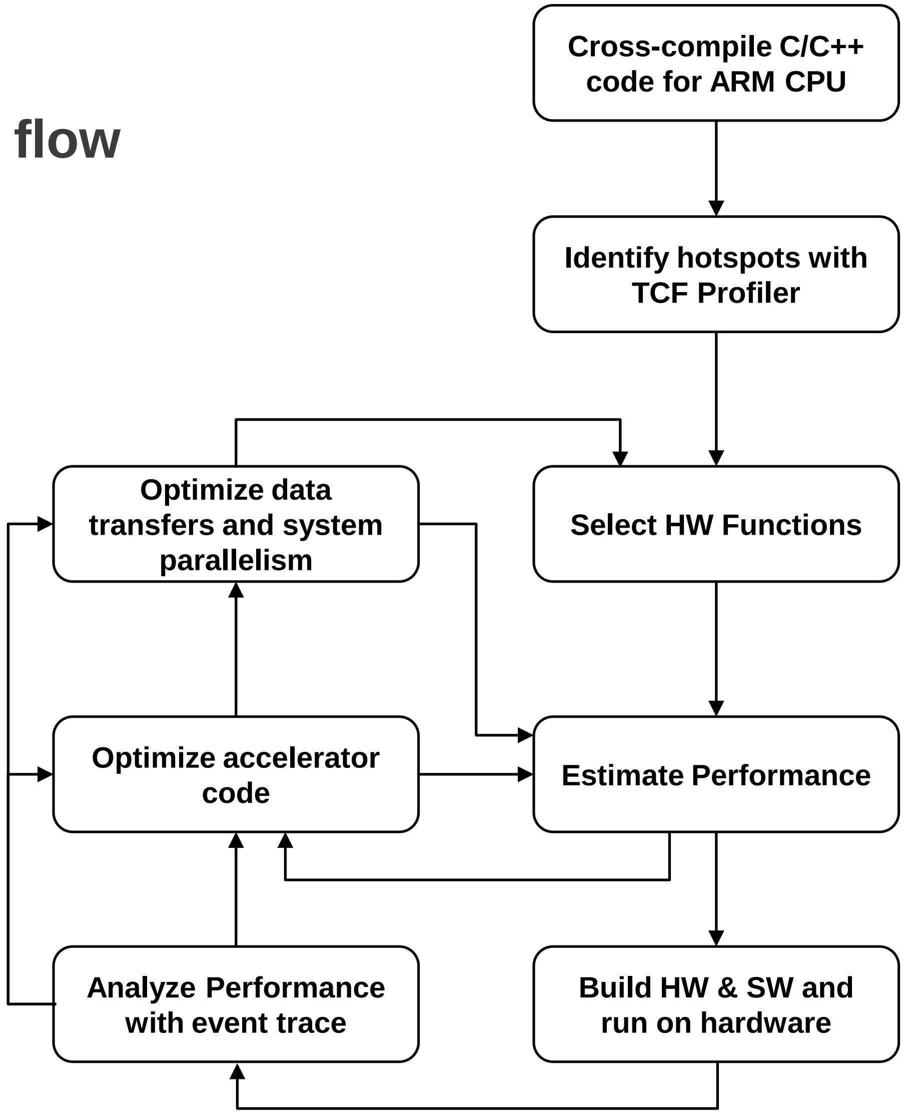


XILINX ALL PROGRAMMABLE.

Agenda

C/C++ to Optimized System

- **Recommended design flow**


Cross-compile C/C++ code for ARM CPU
Identify hotspots with TCF Profiler
Optimize data transfers and system parallelism
Select HW Functions
Optimize accelerator code
Estimate Performance
Analyze Performance with event trace
Build HW & SW and run on hardware

Example : Matrix Multiplication

- **Common compute primitive for many applications, suitable for hardware acceleration**
  - $O(n^3)$ time for schoolbook algorithm, $O(n^{2+\gamma})$ for more sophisticated sequential algorithms
  - Can trade space for time, but must inspect $O(n^2)$ elements in DDR

- **Problem size: 32 x 32 matrices of `float`**
- **Algorithm: schoolbook implementation**

Matrix Multiplication

- **Output element $C_{ij} = A_{i*} \cdot B_{*j}$ (dot product)**

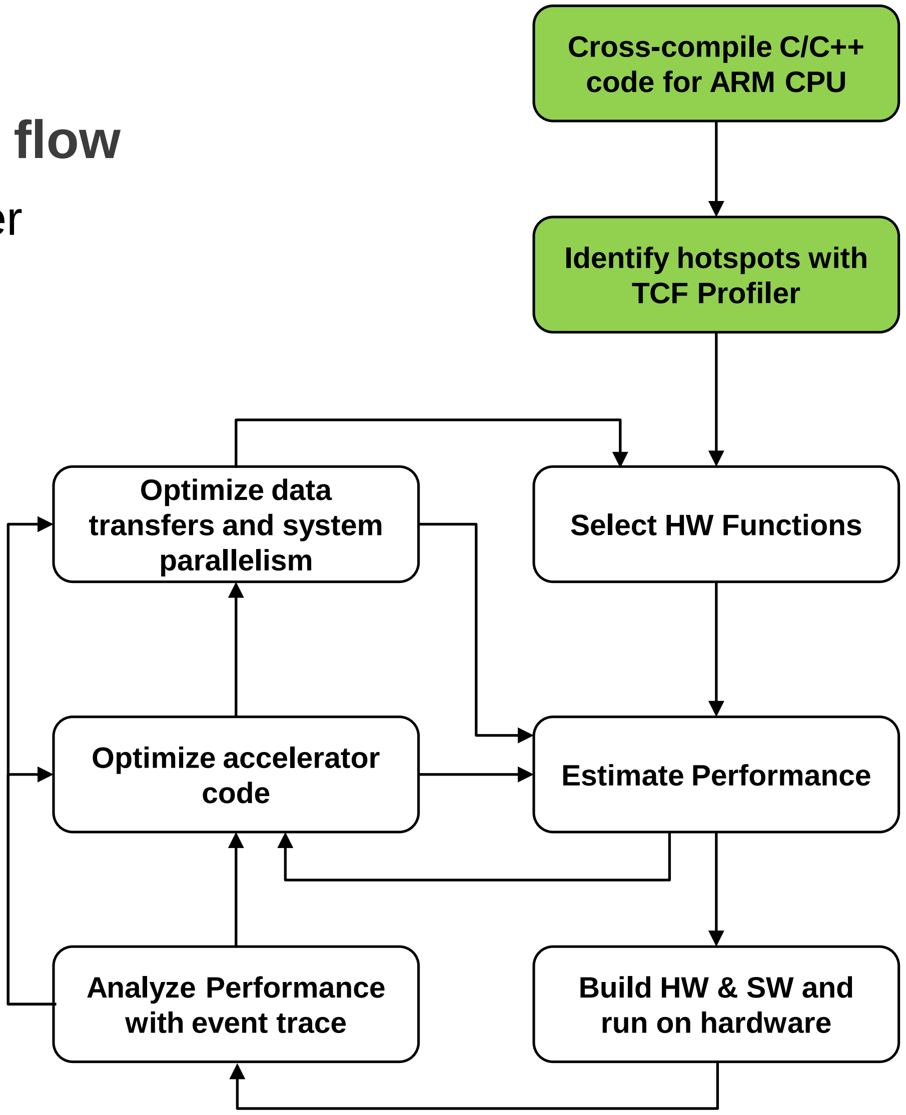
$$\begin{array}{c} \\ \text{row} \\ i=0 \\ i=1 \\ i=2 \end{array}
\begin{array}{c} \underline{\mathbf{A}} \\ \\ \begin{bmatrix} \boxed{11 \quad 12 \quad 13} \\ 14 \quad 15 \quad 16 \\ 17 \quad 18 \quad 19 \end{bmatrix} \end{array}
\;\begin{array}{c} \\ \text{col} \\ \\ * \\ \\ \end{array}\;
\begin{array}{c} \underline{\mathbf{B}} \\ \begin{array}{ccc} j=0 & j=1 & j=2 \end{array} \\ \begin{bmatrix} \boxed{\begin{matrix} 21 \\ 24 \\ 27 \end{matrix}} & \boxed{\begin{matrix} 22 \\ 25 \\ 28 \end{matrix}} & \begin{matrix} 23 \\ 26 \\ 29 \end{matrix} \end{bmatrix} \end{array}
=
\begin{array}{c} \underline{\mathbf{C}} \\ \\ \begin{bmatrix} \boxed{870} & \boxed{906} & 942 \\ 1086 & 1131 & 1176 \\ 1302 & 1356 & 1410 \end{bmatrix} \end{array}$$

$$\begin{aligned} r_{00} &= 11 * 21 \\ &+ 12 * 24 \\ &+ 13 * 27 \\ &= 870 \end{aligned}
\qquad
\begin{aligned} r_{01} &= 11 * 22 \\ &+ 12 * 25 \\ &+ 13 * 28 \\ &= 906 \end{aligned}$$

Etc…

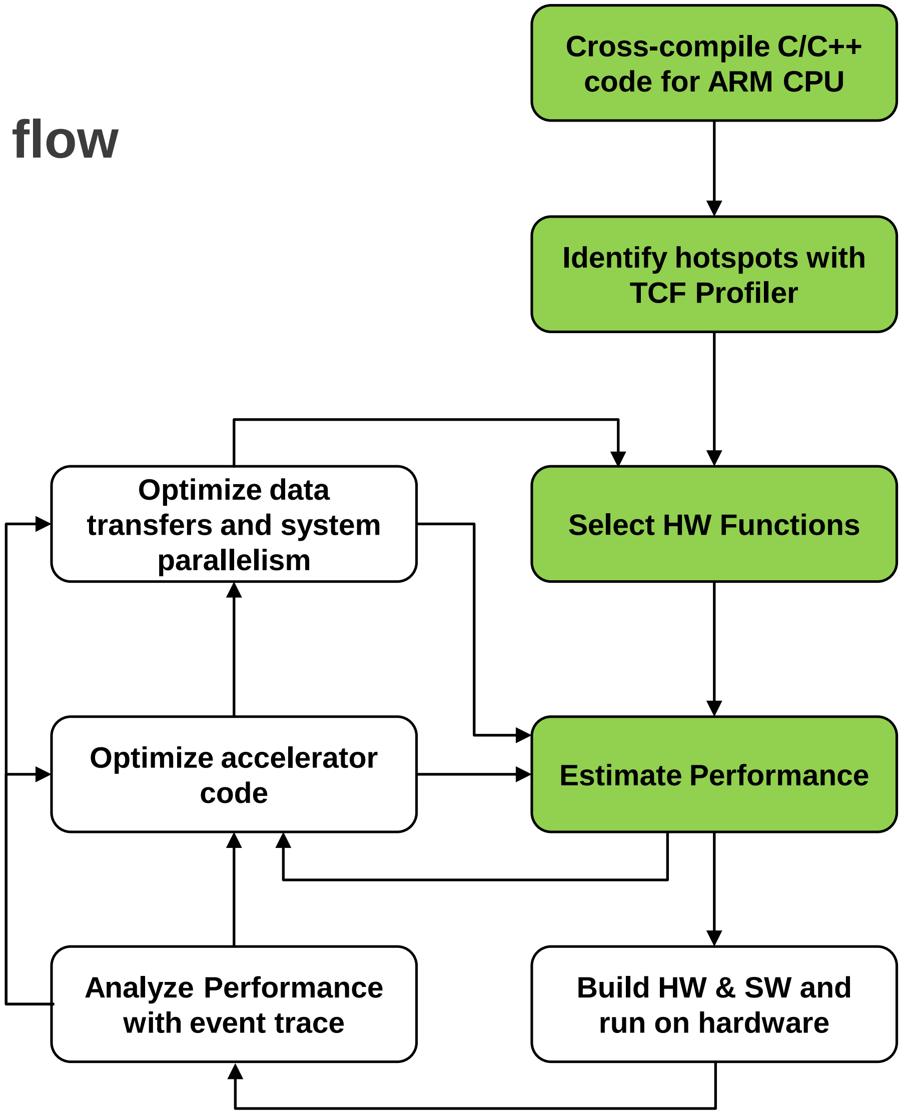
C/C++ to Optimized System

- **Recommended design flow**
  - Profiling with TCF Profiler


Cross-compile C/C++ code for ARM CPU
Identify hotspots with TCF Profiler
Select HW Functions
Optimize data transfers and system parallelism
Optimize accelerator code
Estimate Performance
Analyze Performance with event trace
Build HW & SW and run on hardware

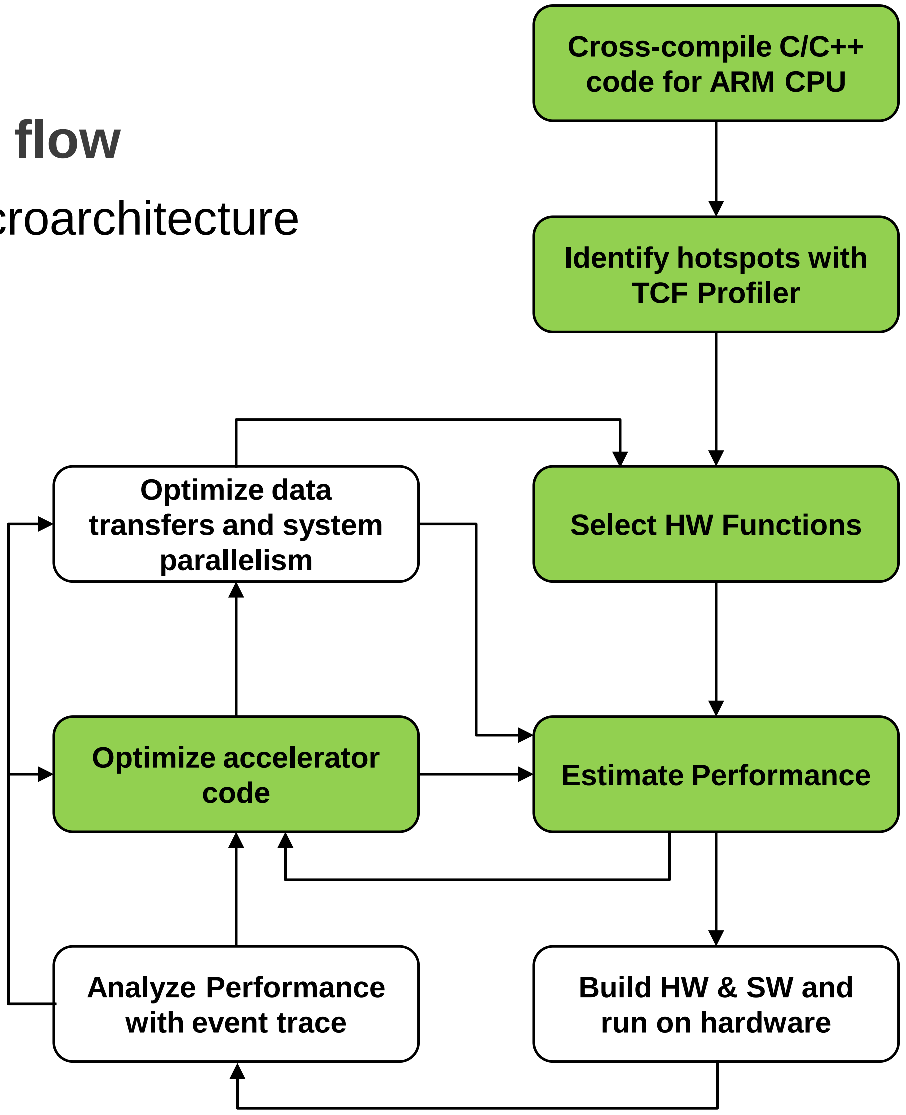
C/C++ to Optimized System

- **Recommended design flow**
  - Performance estimation


Cross-compile C/C++ code for ARM CPU
Identify hotspots with TCF Profiler
Select HW Functions
Optimize data transfers and system parallelism
Estimate Performance
Optimize accelerator code
Analyze Performance with event trace
Build HW & SW and run on hardware

C/C++ to Optimized System

- **Recommended design flow**
  - Optimize accelerator microarchitecture using Vivado HLS


Cross-compile C/C++ code for ARM CPU
Identify hotspots with TCF Profiler
Select HW Functions
Optimize data transfers and system parallelism
Estimate Performance
Optimize accelerator code
Analyze Performance with event trace
Build HW & SW and run on hardware

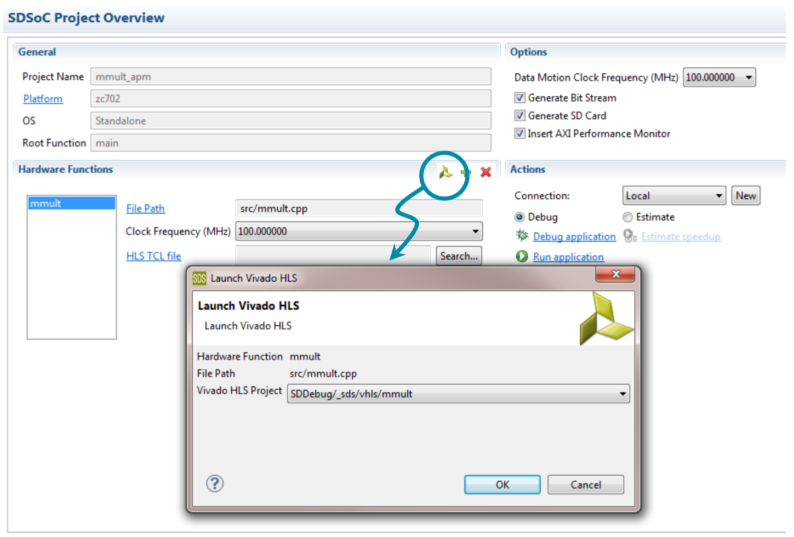
XILINX ALL PROGRAMMABLE.

very brief

A SDSoC programmer's introduction to Vivado HLS

HLS as Cross Compiler

- **SDSoC employs Vivado HLS as programmable logic cross-compiler**
  - Hardware function source code shared between SDSoC and VHLS
    - Requires data type consistency between VHLS and arm-gcc
  - SDSoC automatically creates VHLS projects for synthesized IP blocks
  - User can optionally launch HLS GUI from SDSoC
    - Optimize accelerator code
    - Simulate hardware function



SDSoC Project Overview

General

Project Name: mmult_apm
Platform: zc702
OS: Standalone
Root Function: main

Options

Data Motion Clock Frequency (MHz): 100.000000
[✓] Generate Bit Stream
[✓] Generate SD Card
[✓] Insert AXI Performance Monitor

Hardware Functions

mmult
File Path: src/mmult.cpp
Clock Frequency (MHz): 100.000000
HLS TCL file:

Actions

Connection: Local
[✓ Debug] Estimate

Launch Vivado HLS

Launch Vivado HLS
Launch Vivado HLS

Hardware Function: mmult
File Path: src/mmult.cpp
Vivado HLS Project: SDDebug/_sds/vhls/mmult

Microarchitecture Optimizations

- **The most important HLS compiler directives are familiar to performance-oriented software programmers**

Directives and Configurations	*Description*
PIPELINE	Reduces the initiation interval by allowing the concurrent execution of operations within a loop or function.
DATAFLOW	Enables functions and loops to execute concurrently. Avoid at the top-level hardware function.
INLINE	Inline a function to function hierarchy, enable logic optimization across function boundaries and reduce function call overhead.
UNROLL	Unroll for-loops to create multiple independent operations rather than a single collection of operations.
ARRAY_PARTITION	Partition array into smaller arrays or individual registers to increase concurrent access to data and remove block RAM bottlenecks.

- **Use hardware buffers to improve communication bandwidth between accelerator and external memory**
  – Copy loops at the function boundary when multiple accesses required and to burst data into local buffers

Loop Unrolling and Pipelining

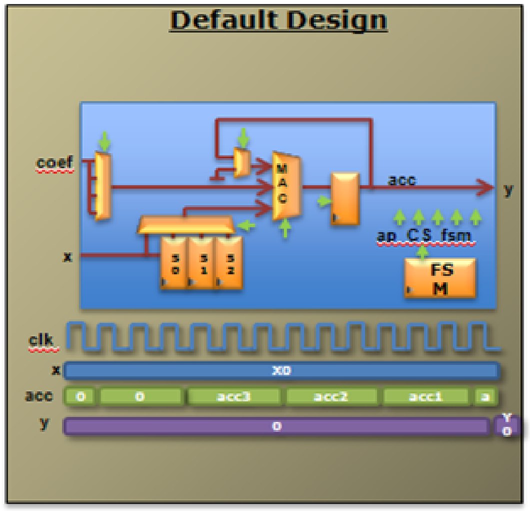
Default Behavior

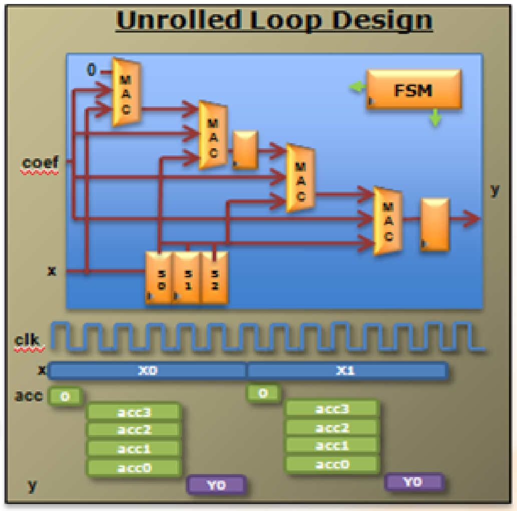
```
...
loop: for (i=3;i>=0;i--) {
  if (i==0) {
    acc+=x*c[0];
    shift_reg[0]=x;
  } else {
     shift_reg[i]=shift_reg[i-1];
     acc+=shift_reg[i]*c[i];
  }
} ....
```

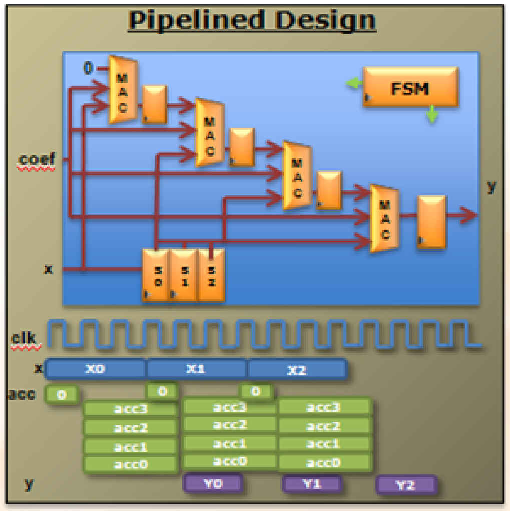
The same hardware is used for each iteration of the loop:
- Small area
- Long latency
- Low throughput

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

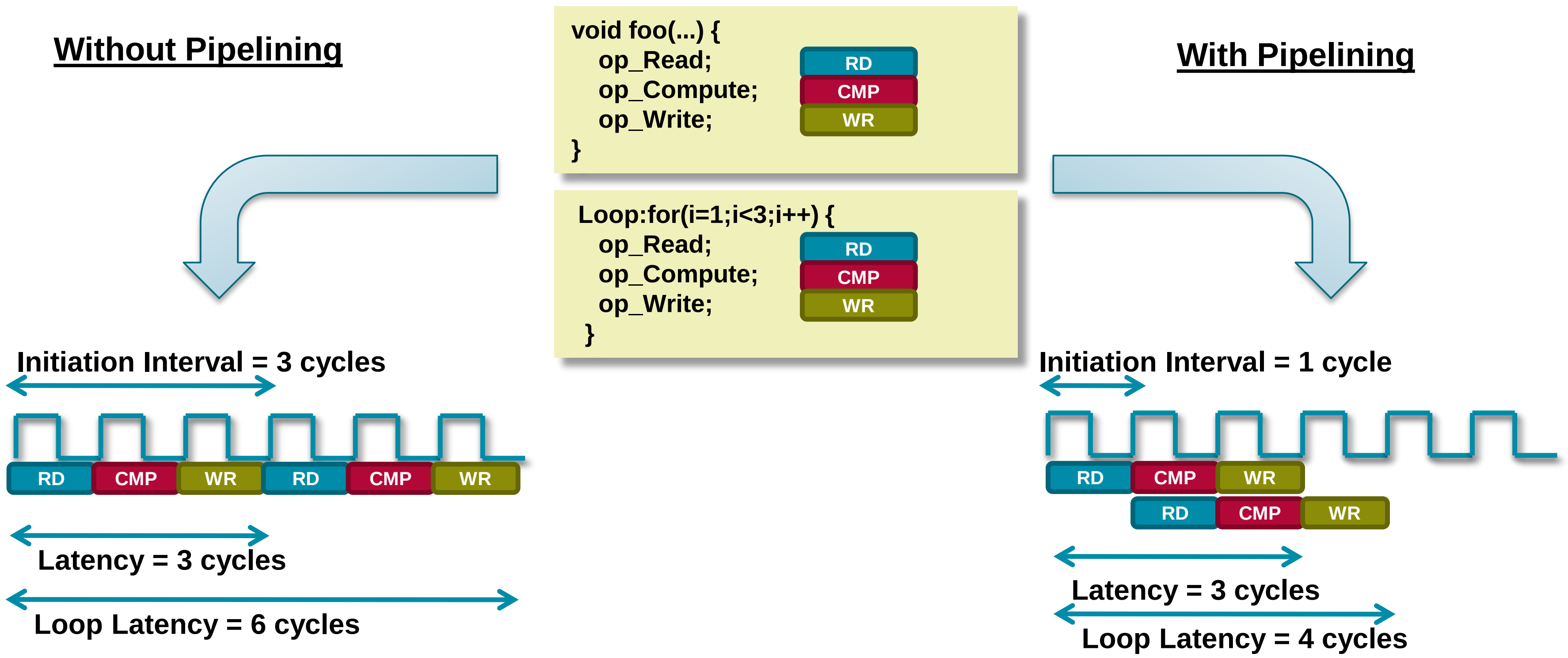
Different iterations are executed concurrently:
- Higher area
- Short latency
- Best throughput







Loop and Function Pipelining

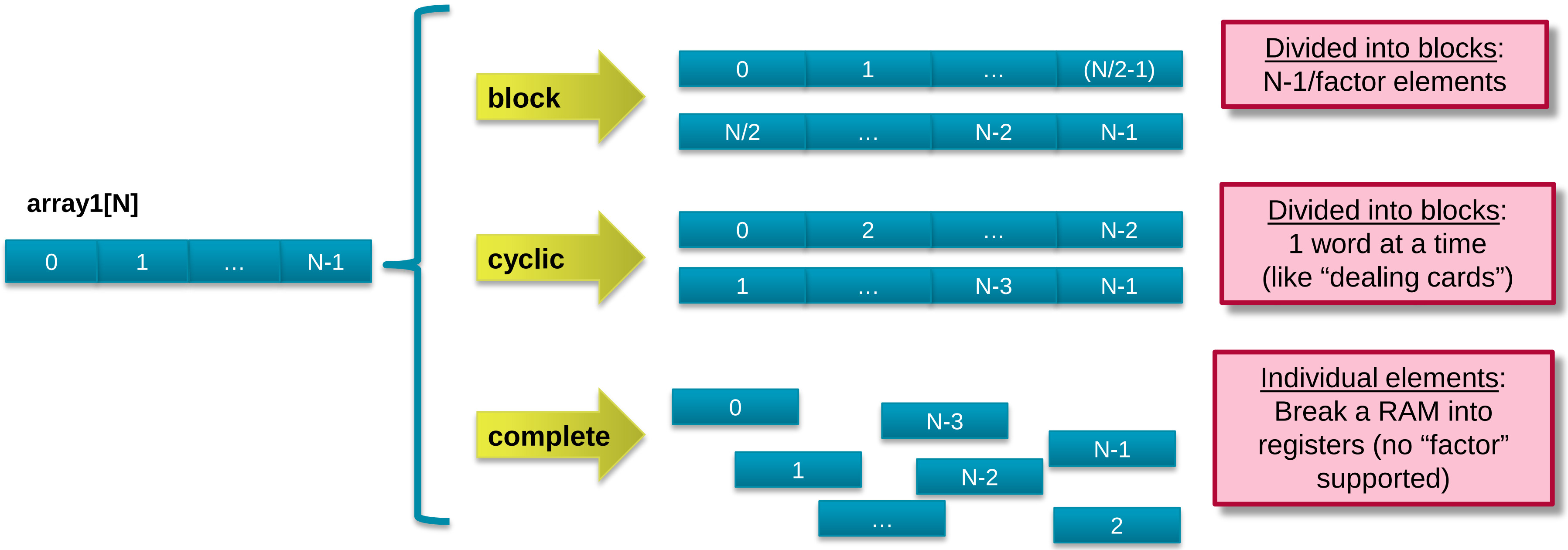


```
for (index_b = 0; index_b < B_NCOLS; index_b++) {
#pragma HLS PIPELINE II=1
     float result = 0;
     for (index_d = 0; index_d < A_NCOLS; index_d++) {
         float product_term = in_A[index_a][index_d] * in_B[index_d][index_b];
         result += product_term;
     }
     out_C[index_a * B_NCOLS + index_b] = result;
}
```

- **Pipelined loops**
  - Combined with array partitioning to achieve II=1

Array Partitioning

- Partition into multiple memories to increase concurrent access



array1[N]
0
1
…
N-1
block
0
1
…
(N/2-1)
N/2
…
N-2
N-1
cyclic
0
2
…
N-2
1
…
N-3
N-1
complete
0
N-3
N-1
1
N-2
…
2
Divided into blocks: N-1/factor elements
Divided into blocks: 1 word at a time (like “dealing cards”)
Individual elements: Break a RAM into registers (no “factor” supported)

```
void mmult_kernel(float in_A[A_NROWS][A_NCOLS], float in_B[A_NCOLS][B_NCOLS], float out_C[A_NROWS*B_NCOLS])
{
#pragma HLS INLINE self
#pragma HLS array_partition variable=in_A block factor=16 dim=2
#pragma HLS array_partition variable=in_B block factor=16 dim=1
// snip
}
```

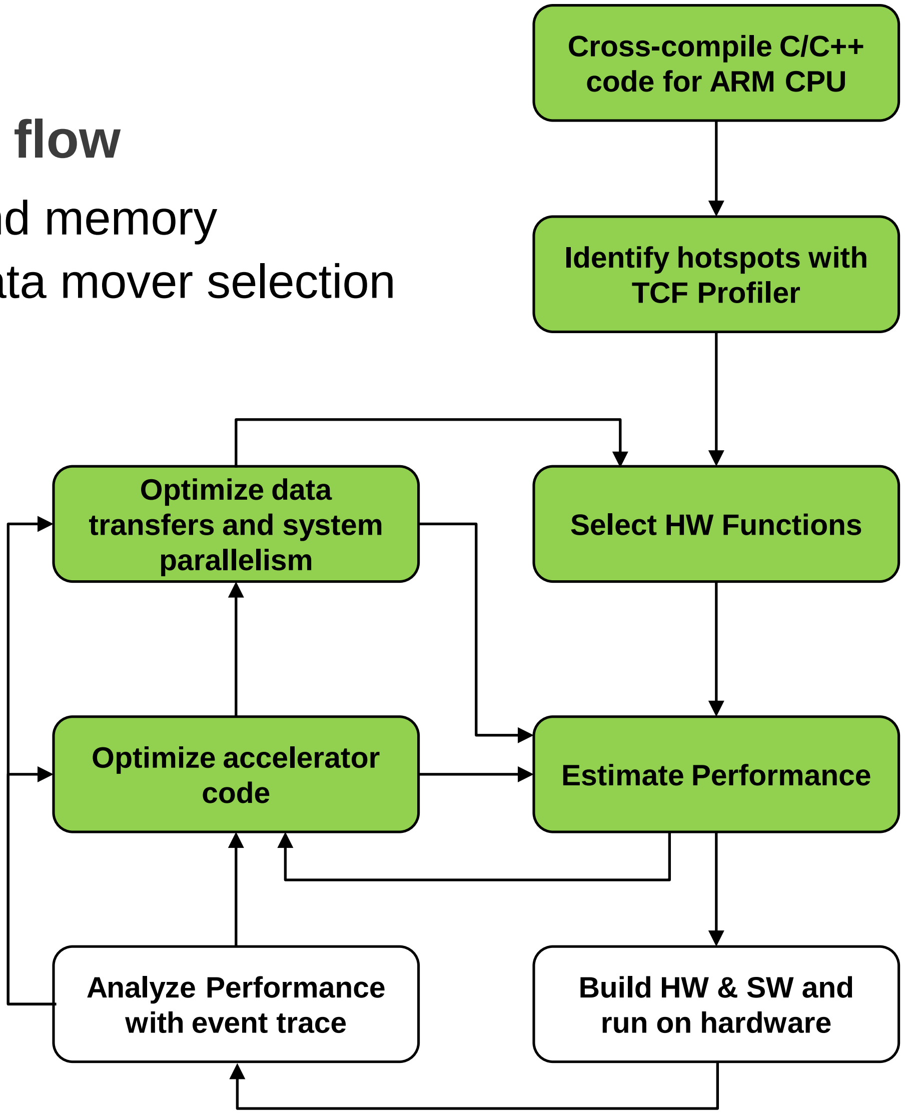
Example: Matrix Multiplication

- **Microarchitecture optimizations**

1. **Pipeline the dot-product loop with II=1 to unroll the inner loop**
2. **Add pipelined copy loops to local dual-port BRAMs partitioned for parallel access**

C/C++ to Optimized System

- **Recommended design flow**
  - – Use SDSoC pragmas and memory allocation to influence data mover selection


Cross-compile C/C++ code for ARM CPU
Identify hotspots with TCF Profiler
Optimize data transfers and system parallelism
Select HW Functions
Optimize accelerator code
Estimate Performance
Analyze Performance with event trace
Build HW & SW and run on hardware

XILINX ALL PROGRAMMABLE.

System optimizations

- **Data mover inference based on program properties**
  - Transfer size
  - Memory properties: physical contiguity
  - Accelerator memory access patterns

- **Platform interface connectivity based on program properties**
  - Transfer size
  - Memory properties: cacheability

- **Performance bottlenecks to avoid**
  - Pointer arithmetic is usually ill-suited for hardware
    - Instead, burst chunks of data into FIFOs or BRAM for accelerator access
  - Transferring data through cache when CPU doesn't touch it
  - Transferring cacheable memory through HP ports

Example: Matrix Multiplication

- **Microarchitecture optimizations**

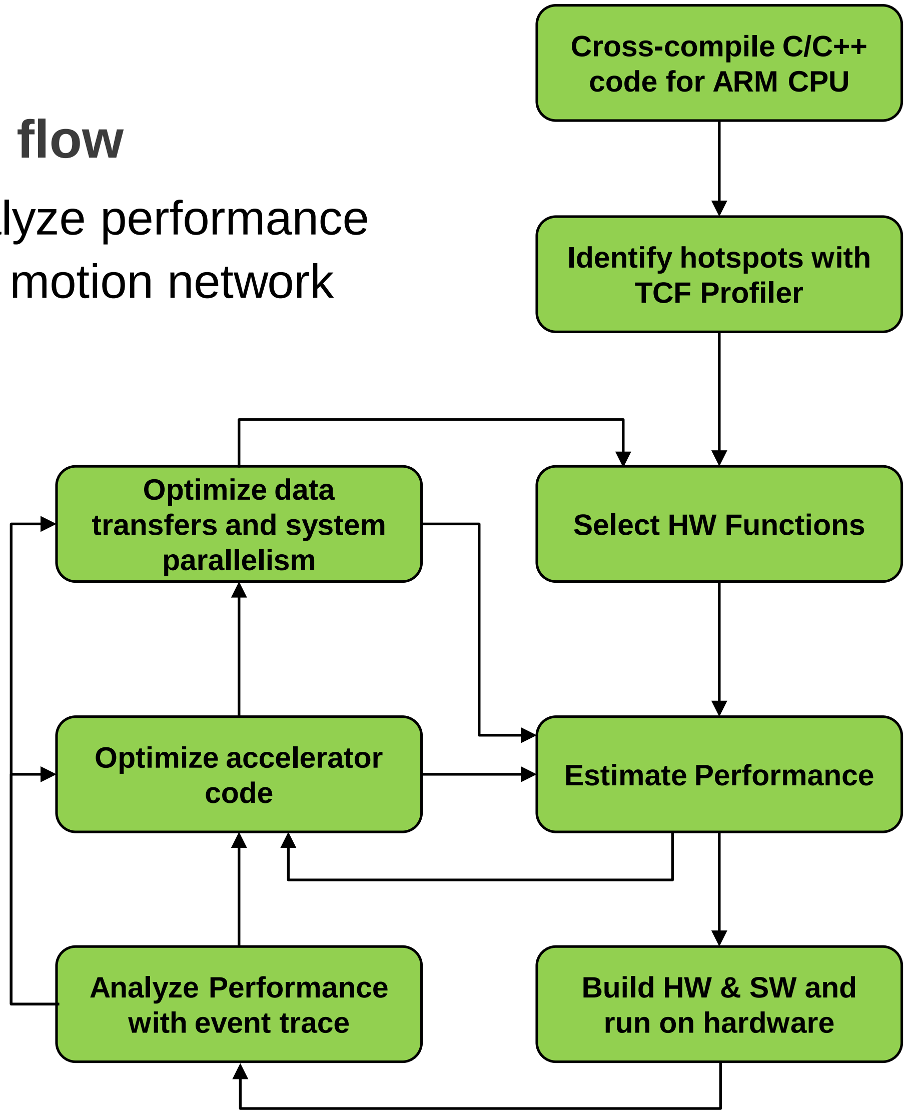
1. **Pipeline the dot-product loop with II=1 to unroll the inner loop**
2. **Add pipelined copy loops to local dual-port BRAMs partitioned for parallel access**

- **System optimizations**

1. **Sequential access pragma**
2. **Allocate buffers in physically contiguous memory for most efficient DMA (`axidma_simple`)**

C/C++ to Optimized System

- **Recommended design flow**
  - Use event tracing to analyze performance of accelerators and data motion network


Cross-compile C/C++ code for ARM CPU
Identify hotspots with TCF Profiler
Select HW Functions
Optimize data transfers and system parallelism
Optimize accelerator code
Estimate Performance
Analyze Performance with event trace
Build HW & SW and run on hardware

HW/SW Event Tracing

- **Automatic software and hardware instrumentation for performance monitoring**
- **Provide visibility into “higher level events” during program execution, with finer granularity than overall run time**
  – Accelerator tasks
  – Data transfers between accelerators and between accelerators and PS
- **Assist in system debugging, showing “what happened when”**
- **Provide application-specific trace points**
  – e.g., depending on accelerators
- **Minimize impact on execution time and PL area**

Trace Example

Matrix Multiplication

Original Code

main function

```
int main(int argc, char* argv[]) {
   float *A, *B, *C;

   init(A, B, C);
   mmult(A, B, C);

   check(C);
}
```

mmult function

```
void mmult(float *A, float *B, float *C) {
   for (int a=0; a<A_NROWS; a++)
      for (int b=0; b<B_NCOLS; b++) {
         float result = 0;
         for (int c=0; c<A_NCOLS; c++)
            result += A[a][c]*B[c][b];
         C[a][b] = result;
      }
}
```

Trace Example

Matrix Multiplication

main function

mmult function

Original Code

```
int main(int argc, char* argv[]) {
   float *A, *B, *C;

   init(A, B, C);
   mmult(A, B, C);

   check(C);
}
```

```
void mmult(float *A, float *B, float *C) {
   for (int a=0; a<A_NROWS; a++)
      for (int b=0; b<B_NCOLS; b++) {
         float result = 0;
         for (int c=0; c<A_NCOLS; c++)
            result += A[a][c]*B[c][b];
         C[a][b] = result;
      }
}
```

Stub Code

```
int main(int argc, char* argv[]) {
   float *A, *B, *C;

   init(A, B, C);
   _p0_mmult_0(A, B, C);

   check(C);
}
```

```
void _p0_mmult_0(float *A, float *B, float *C) {
   cf_send_i(&req0, cmd);
   cf_wait(req0);

   cf_send_i(&req1, A);
   cf_send_i(&req2, B);
   cf_wait(req1);
   cf_wait(req2);
}
```

Trace Example

- **Matrix Multiplication**

Original Code

main function

```
int main(int argc, char* argv[]) {
   float *A, *B, *C;

   init(A, B, C);
   mmult(A, B, C);

   check(C);
}
```

mmult function

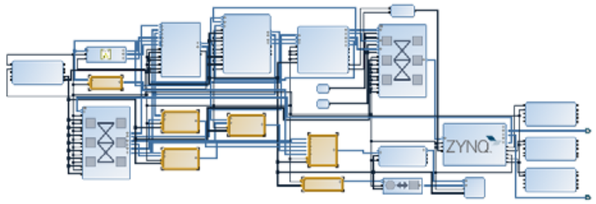
```
void mmult(float *A, float *B, float *C){
   for (int a=0; a<A_NROWS; a++)
      for (int b=0; b<B_NCOLS; b++) {
         float result = 0;
         for (int c=0; c<A_NCOLS; c++)
            result += A[a][c]*B[c][b];
         C[a][b] = result;
      }
}
```

Trace Code

```
int main(int argc, char* argv[]) {
   float *A, *B, *C;

   init(A, B, C);
   _p0_mmult_0(A, B, C);

   check(C);
}
```

No change

```
void _p0_mmult_0(float *A, float *B, float *C) {
   sds_trace(EVENT_START);
   cf_send_i(&req0, cmd);
   sds_trace(EVENT_STOP);
   sds_trace(EVENT_START);
   cf_wait(req0);
   sds_trace(EVENT_STOP);
   sds_trace(EVENT_START);
   cf_send_i(&req1, A);
   sds_trace(EVENT_STOP);
   sds_trace(EVENT_START);
   cf_send_i(&req2, B);
   sds_trace(EVENT_STOP);
   sds_trace(EVENT_START);
   cf_wait(req1);
   sds_trace(EVENT_STOP);
   sds_trace(EVENT_START);
   cf_wait(req2);
   sds_trace(EVENT_STOP);
   sds_trace(EVENT_START);
}
```



Added IPs
- 4 AXI Stream monitors
- 1 Accelerator monitor
- Trace output infrastructure

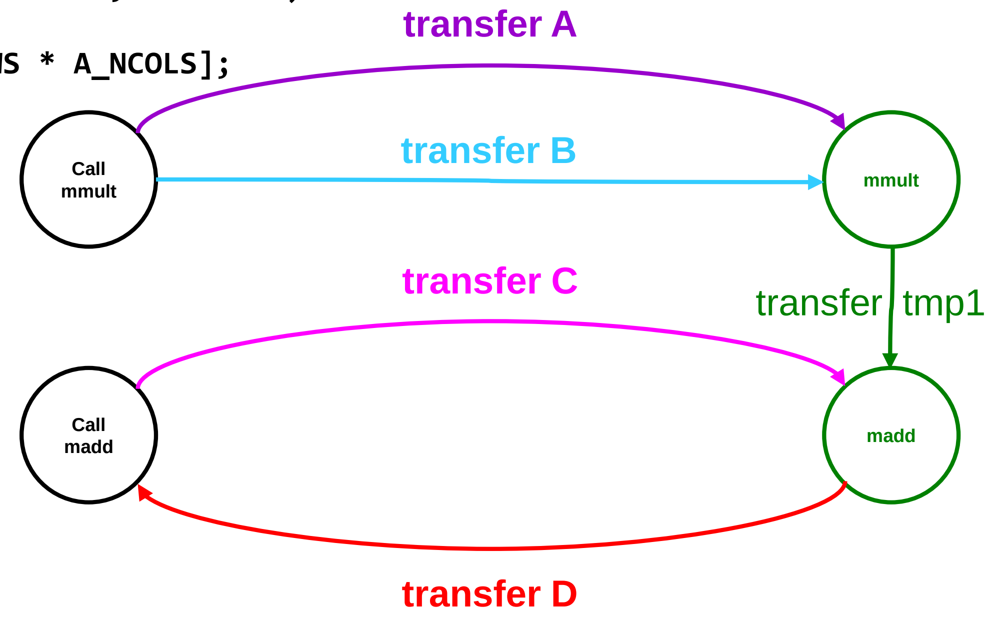
Example: Matrix Multiply-Add

- **Add matrix addition operator to demonstrate how to construct hardware pipelines to increase concurrent computation**

- **SDSoC compiler will create hardware 'direct connections' between accelerators and between platform and accelerators**
  - Program dataflow analysis to ensure correct behavior
  - Software synchronization automatically instrumented by the compiler

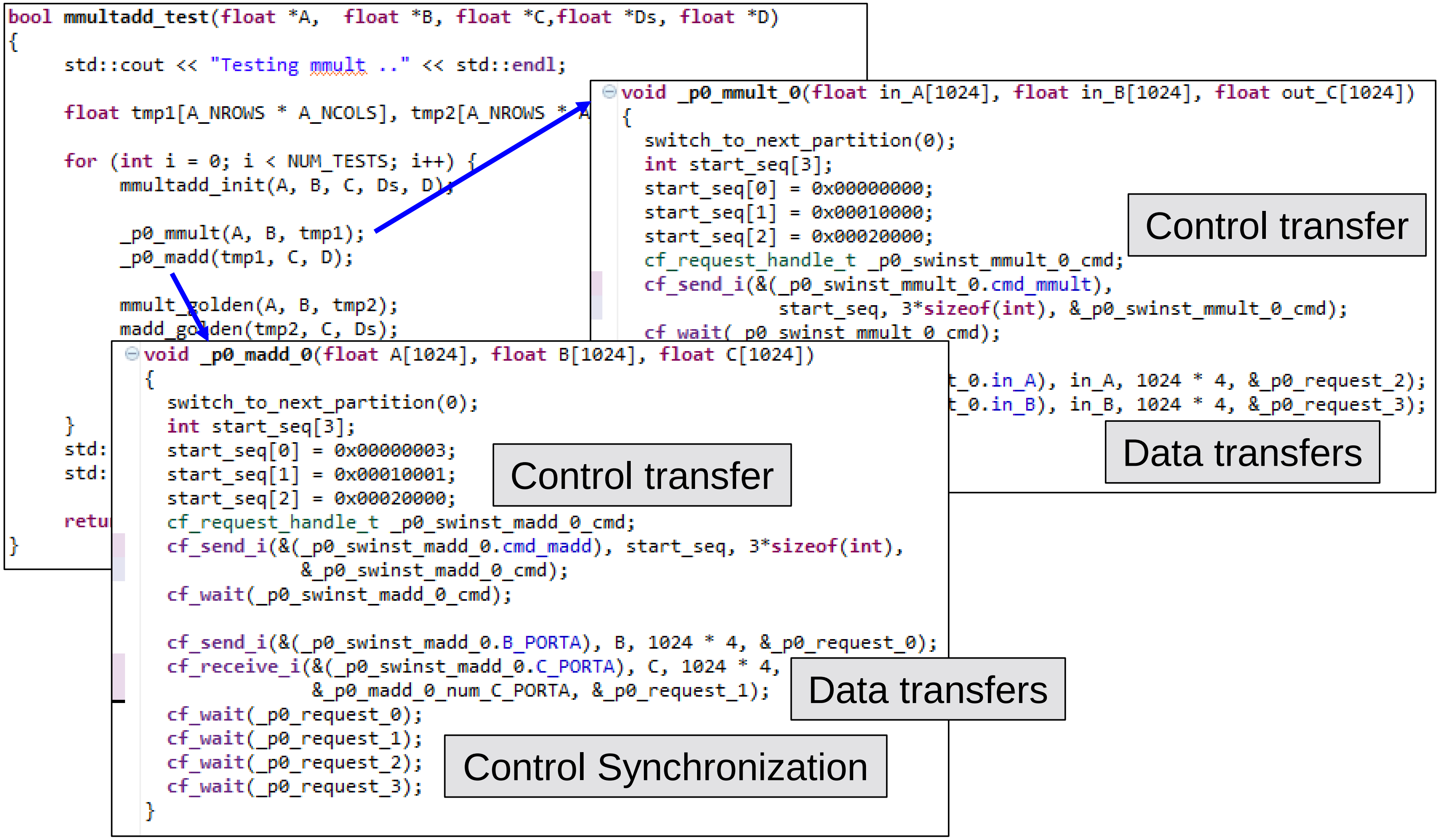
How SDSoC Compiler Maps Programs to HW/SW

```
bool mmultadd(float *A,  float *B, float *C,float *Ds, float *D)
{
     float tmp1[A_NROWS * A_NCOLS], tmp2[A_NROWS * A_NCOLS];

     for (int i = 0; i < NUM_TESTS; i++) {
          mmultadd_init(A, B, C, Ds, D);

          mmult(A, B, tmp1);
          madd(tmp1, C, D);

          mmult_golden(A, B, tmp2);
          madd_golden(tmp2, C, Ds);

          if (!mmult_result_check(D, Ds))
               return false;
     }
     return true;
}
```



How SDSoC Compiler Maps Programs to SW

Structure of generated software



```
bool mmultadd_test(float *A,  float *B, float *C,float *Ds, float *D)
{
     std::cout << "Testing mmult .." << std::endl;

     float tmp1[A_NROWS * A_NCOLS], tmp2[A_NROWS * A

     for (int i = 0; i < NUM_TESTS; i++) {
          mmultadd_init(A, B, C, Ds, D);

          _p0_mmult(A, B, tmp1);
          _p0_madd(tmp1, C, D);

          mmult_golden(A, B, tmp2);
          madd_golden(tmp2, C, Ds);

     }
     std:
     std:

     retu
}
```

```
void _p0_mmult_0(float in_A[1024], float in_B[1024], float out_C[1024])
{
  switch_to_next_partition(0);
  int start_seq[3];
  start_seq[0] = 0x00000000;
  start_seq[1] = 0x00010000;
  start_seq[2] = 0x00020000;
  cf_request_handle_t _p0_swinst_mmult_0_cmd;
  cf_send_i(&(_p0_swinst_mmult_0.cmd_mmult),
              start_seq, 3*sizeof(int), &_p0_swinst_mmult_0_cmd);
  cf_wait(_p0_swinst_mmult_0_cmd);

t_0.in_A), in_A, 1024 * 4, &_p0_request_2);
t_0.in_B), in_B, 1024 * 4, &_p0_request_3);
```

```
void _p0_madd_0(float A[1024], float B[1024], float C[1024])
{
  switch_to_next_partition(0);
  int start_seq[3];
  start_seq[0] = 0x00000003;
  start_seq[1] = 0x00010001;
  start_seq[2] = 0x00020000;
  cf_request_handle_t _p0_swinst_madd_0_cmd;
  cf_send_i(&(_p0_swinst_madd_0.cmd_madd), start_seq, 3*sizeof(int),
            &_p0_swinst_madd_0_cmd);
  cf_wait(_p0_swinst_madd_0_cmd);

  cf_send_i(&(_p0_swinst_madd_0.B_PORTA), B, 1024 * 4, &_p0_request_0);
  cf_receive_i(&(_p0_swinst_madd_0.C_PORTA), C, 1024 * 4,
             &_p0_madd_0_num_C_PORTA, &_p0_request_1);
  cf_wait(_p0_request_0);
  cf_wait(_p0_request_1);
  cf_wait(_p0_request_2);
  cf_wait(_p0_request_3);
}
```

Summary

- **System performance achieved through accelerator and system level optimizations**
  - SDSoC compiler creates function pipelines with direct connections in hardware

- **Increase concurrency within accelerators using HLS directives**
  - Pipeline and dataflow loops, function calls, and operations
  - Copy data samples into local BRAM to improve burst read/write and partition to increase compute / memory bandwidth within accelerator
  - UG902: HLS User Guide for more details

- **Data mover and system connectivity inference from user program**
  - Data mover selection based on buffer allocation, transfer payload
  - System connections and driver efficiency based on program memory properties, e.g., cacheability
  - UG1027: SDSoC User Guide for more details

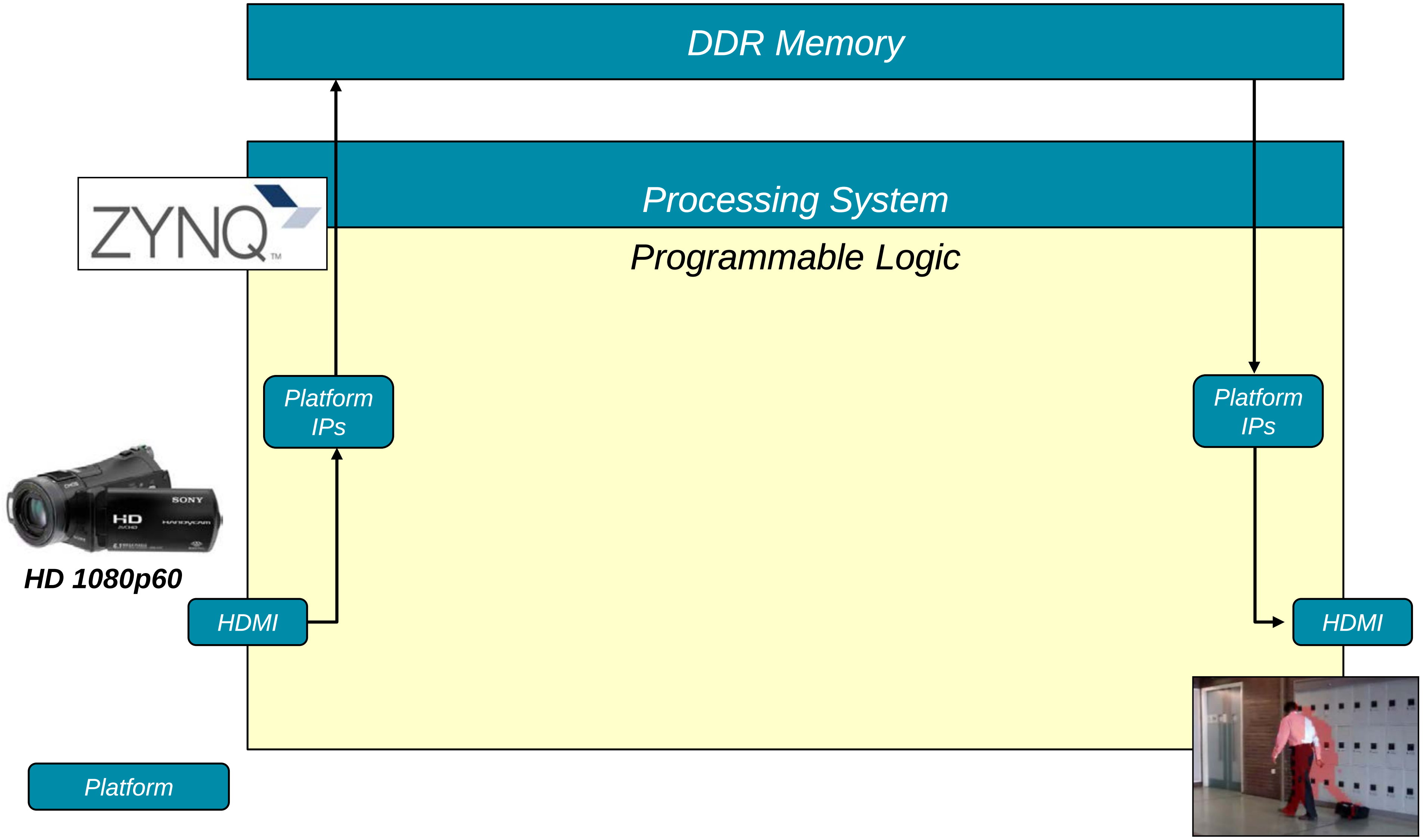
Agenda

- Zynq SoC and MPSoC Architecture
- SDSoC Overview
- Real-life Success
- C/C++ to Optimized System
- **Targeting Your Own Platform**
- Next Steps

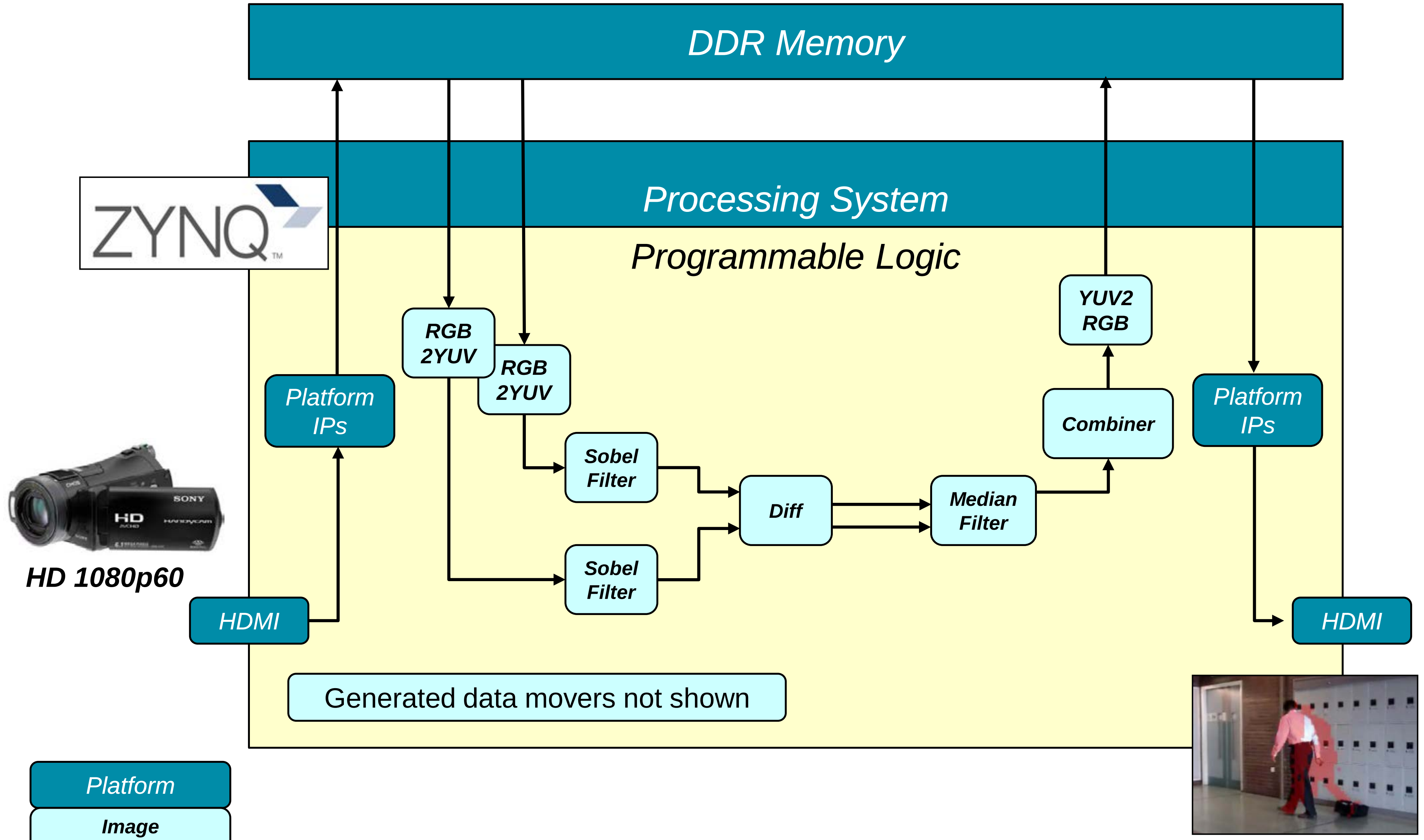
Platform-Based Design

- **We make a clear distinction between *platforms* and *Software-Defined systems on chip***
- **A *platform* is a base system designed for reuse**
  - Processing system, I/O subsystems, memory interfaces,…
  - OS, device drivers, boot loaders, file system, libraries,…
  - Built using standard SoC HW & SW design methodologies and tools
- **A *software-defined SoC* extends a platform with application-specific hardware and software**
  - User specifies functions for programmable logic
  - Compiler analyzes program and compiles into an application-specific SoC
  - Hardware accelerator and data motion network
  - **`#pragmas to`** assist and override compiler

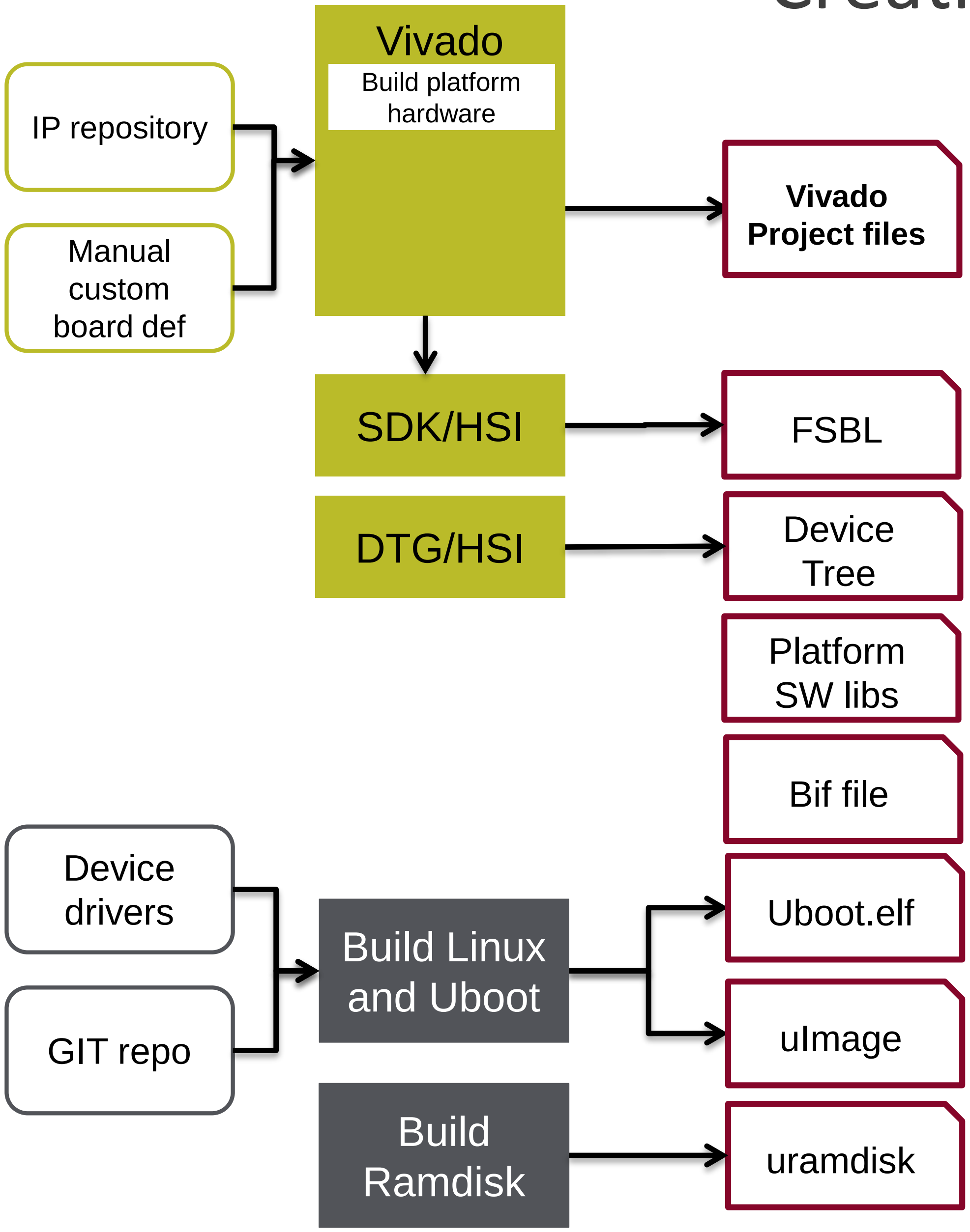
zc702_trd Platform (Targeted Reference Design)

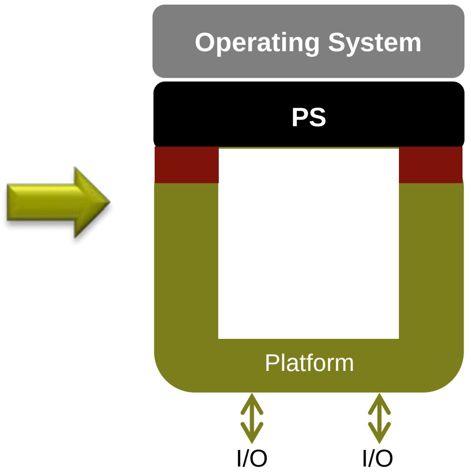


Motion Detection Application SoC



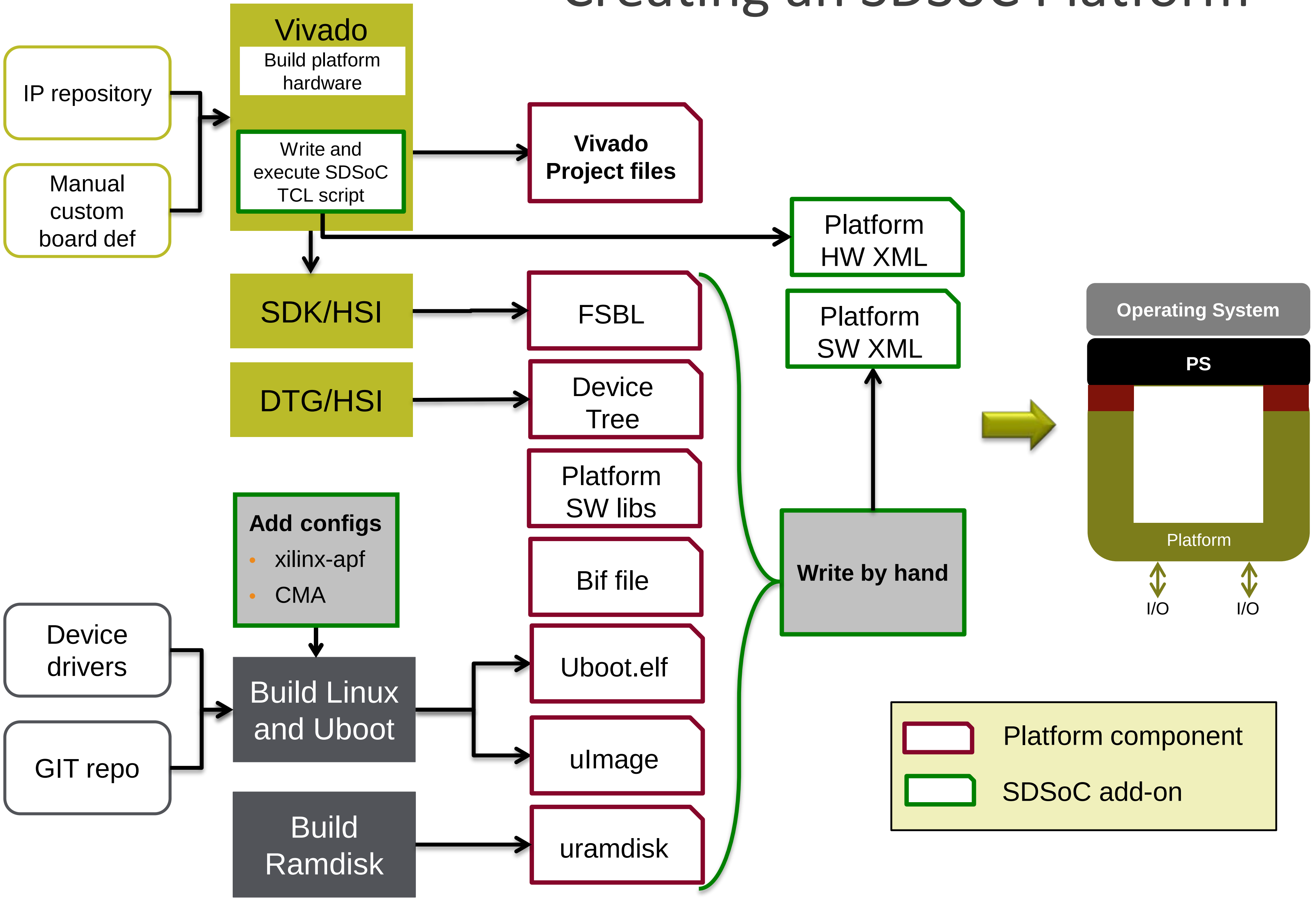
Creating a Platform





▭	Platform component

Creating an SDSoC Platform



SDSoC Platform Hardware

- **Start from essentially any Vivado hardware system**
  - Zynq-7000® or Zynq-UltraSCALE+ MPSoC® processing system
  - Memory interfaces, custom I/O, and other peripherals
  - Set of AXI, AXI-S, clocks, resets, interrupt ports
- **Create TCL script**
  - Declare platform interfaces in a Vivado block diagram
  - Generate platform hardware description XML file

SDSoC Platform Hardware APIs

- zc702

```
set pfm [sdsoc::create_pfm zc702_hw.pfm]
sdsoc::pfm_name        $pfm "xilinx.com" "xd" "zc702" "1.0"
sdsoc::pfm_description $pfm "Zynq ZC702 Board"
sdsoc::pfm_clock       $pfm FCLK_CLK0 ps7 0 false proc_sys_reset_0
sdsoc::pfm_clock       $pfm FCLK_CLK1 ps7 1 false proc_sys_reset_1
sdsoc::pfm_clock       $pfm FCLK_CLK2 ps7 2 true  proc_sys_reset_2
sdsoc::pfm_clock       $pfm FCLK_CLK3 ps7 3 false proc_sys_reset_3
sdsoc::pfm_axi_port    $pfm M_AXI_GP0 ps7 M_AXI_GP
sdsoc::pfm_axi_port    $pfm M_AXI_GP1 ps7 M_AXI_GP
sdsoc::pfm_axi_port    $pfm S_AXI_ACP ps7 S_AXI_ACP
sdsoc::pfm_axi_port    $pfm S_AXI_HP0 ps7 S_AXI_HP
sdsoc::pfm_axi_port    $pfm S_AXI_HP1 ps7 S_AXI_HP
sdsoc::pfm_axi_port    $pfm S_AXI_HP2 ps7 S_AXI_HP
sdsoc::pfm_axi_port    $pfm S_AXI_HP3 ps7 S_AXI_HP
for {set i 0} {$i < 16} {incr i} {
  sdsoc::pfm_irq       $pfm In$i xlconcat
}
sdsoc::generate_hw_pfm $pfm
```

SDSoC Platform Software

- **Operating systems**
  - Linux, bare metal, FreeRTOS
- **Boot loaders**
  - FSBL, U-Boot
- **Library files**
  - Needed for cross-compiling and linking application code
  - Shared libraries must also reside in target rootfs

- **Platform software description metadata file**
  - Provides information needed to compile, link, generate SD cards, etc.
  - Written by hand by platform provider

SDSoC Platform Software Description

- **zc702**

```
 <xd:bootFiles
      xd:os="linux"
      xd:bif="boot/linux.bif"
      xd:readme="boot/generic.readme"
      xd:devicetree="boot/devicetree.dtb"
      xd:linuxImage="boot/uImage"
      xd:ramdisk="boot/uramdisk.image.gz"
/>

<xd:bootFiles
      xd:os="standalone"
      xd:bif="boot/standalone.bif"
      xd:readme="boot/generic.readme"
/>

<xd:bootFiles
      xd:os="freertos"
      xd:bif="boot/freertos.bif"
      xd:readme="boot/generic.readme"
/>
```

```
<xd:libraryFiles
      xd:os="standalone"
      xd:libDir="arm-xilinx-eabi/lib"
      xd:ldscript="arm-xilinx-eabi/lscript.ld"
/>

<xd:libraryFiles
      xd:os="freertos"
      xd:osDepend="standalone"
      xd:includeDir="/arm-xilinx-eabi/include/freertos"
      xd:libDir="/arm-xilinx-eabi/lib/freertos"
      xd:libName="freertos"
      xd:ldscript="freertos/lscript.ld"
/>
```

```
  <xd:hardware
       xd:system="prebuilt"
       xd:bitstream="hardware/prebuilt/bitstream.bit"
       xd:export="hardware/prebuilt/export"
       xd:swcf="hardware/prebuilt/swcf"
       xd:hwcf="hardware/prebuilt/hwcf"
     />
```

Testing Your SDSoC Platform

Platform checklist with design guidelines

Reference: ug1146 SDSoC Platforms and Libraries
Location: `<sdsoc_root>/docs`

	Description	Notes		Examples
	Vivado design			
	PS configuration	- Interrupts enabled and connected to xlconcat block that is exported in the platform interface - IRQs associated with available interrupts in platform_hw.pfm - At least one M_AXI_GP port exported as part of platform interface - Any "shared" PS AXI port must have an attached axi_interconnect that exports at least one unused master (for PS slave) or slave (for PS master) port		- zc702_acp, zc706_mem
	Non-PS7 AXI interfaces	- any exported AXI-S interface must have TLAST, TKEEP sideband signals - any exported AXI-S interface must be drivable by the SDSoC clock (dmclkid), i.e., clock-domain-crossing must be handled within the platform - each AXI-S interface must have associated platform library function as described in UG1146		- zc702_axis_io
	Clocks	- Each exported clock listed in platform_hw.pfm - Default clock defined - Each exported clock has associated proc_sys_reset		- zc702, zc706, zed, microzed, microzed20, zybo
	Resets	- Each exported platform clock has proc_sys_reset - Each proc_sys_reset exports peripheral_areset, peripheral_aresetn, interconnect_aresetn		- zc702, zc706, zed, microzed, microzed20, zybo
	Directory structure	- Top directory name must be platform name		

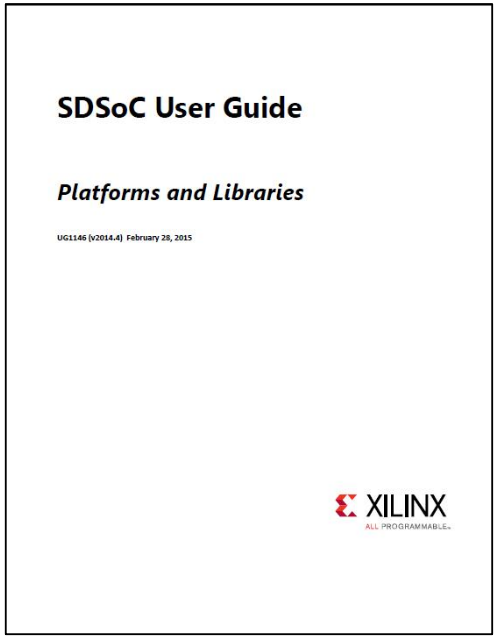
And basic datamover conformance tests

Basic platform tests to ensure that all SDSoC datamovers work on platform
In addition, provide a test for every external I/O function

platform_dm_test.zip

Each of these these tests should run on any platform containing the corresponding Zynq® device		Each custom I/O should have at least one test and sample design for users		
Description	**Datamovers**	**make command for reference test**	**Reference platforms**	**Zynq® device**
axi_dma_simple datamover test	axi_lite, axi_dma_simple	`make axidma_simple [PLATFORM=<your_platform>]`	zc702, zc706, zed, microzed20, microzed, zybo	xc7z010, xc7z015, xc7z020, xc7z030, xc7z035, xc7z045, xc7z100
axi_dma_sg datamover test	axi_lite, axi_dma_sg	`make axidma_sg [PLATFORM=<your_platform>]`	zc702, zc706, zed, microzed20, microzed, zybo	xc7z010, xc7z015, xc7z020, xc7z030, xc7z035, xc7z045, xc7z100
axidma_2d datamover test	axi_lite, axi_dma_2d	`make axidma_2d [PLATFORM=<your_platform>]`	zc702, zc706, zed, microzed20, microzed, zybo	xc7z010, xc7z015, xc7z020, xc7z030, xc7z035, xc7z045, xc7z100
axi_fifo datamover test	axi_lite, axi_fifo	`make axififo [PLATFORM=<your_platform>]`	zc702, zc706, zed, microzed20, microzed, zybo	xc7z010, xc7z015, xc7z020, xc7z030, xc7z035, xc7z045, xc7z100
zero_copy datamover test	axi_lite, zero_copy	`make zero_copy [PLATFORM=<your_platform>]`	zc702, zc706, zed, microzed20, microzed, zybo	xc7z010, xc7z015, xc7z020, xc7z030, xc7z035, xc7z045, xc7z100
axis_accelerator_adapter test	axi_lite, axidma_simple	`make xd_adapter [PLATFORM=<your_platform>]`	zc702, zc706, zed, microzed20, microzed, zybo	xc7z010, xc7z015, xc7z020, xc7z030, xc7z035, xc7z045, xc7z100

Available SDSoC Platforms


SDSoC User Guide

Platforms and Libraries

UG1146 (v2014.4) February 28, 2015

- **Standard "memory-based I/O" platforms**
  - zc702, zc706, zed, zybo, microzed
- **Video & image processing oriented platforms**
  - zc702_trd, zc706_trd (separate download)
  - zc702_osd, zed_osd
- **Additional downloads from Xilinx and partners**
  - Zynq base targeted reference designs (zc702_trd, zc706_trd)
  - http://www.xilinx.com/products/design-tools/software-zone/sdsoc.html#boardskits
- **Teaching platform examples**
  - zc702_axis_io – direct I/O
  - zc702_led – software control of platform IPs (standalone, Linux)
  - zc702_acp – sharing an AXI interface between platform and sdscc

Summary

- **Platform-based design increases productivity and encourages design reuse**
  - Many applications can target a single platform
  - An application can target multiple platforms

- **SDSoC platforms are simple extensions of standard hardware / software systems that enable design reuse**
  - Hardware platform easily exported from Vivado
  - Software platform built using standard flows, simple metadata file

Agenda

- Zynq SoC and MPSoC Architecture
- SDSoC Overview
- Real-life Success
- C/C++ to Optimized System
- Targeting Your Own Platform
- **Next Steps**

Next Steps

- **Hands-on training with one of our Authorized Training Providers**
- **Video Tutorials**
- **User Guides**

- **To further enhance your productivity, consider:**
  - – Libraries & Design Examples
  - – Boards, Kits & Modules

Self-Training Material

Video Tutorials

- Custom Platform Creation
- Estimation & Implementation
- Optimization & Debug

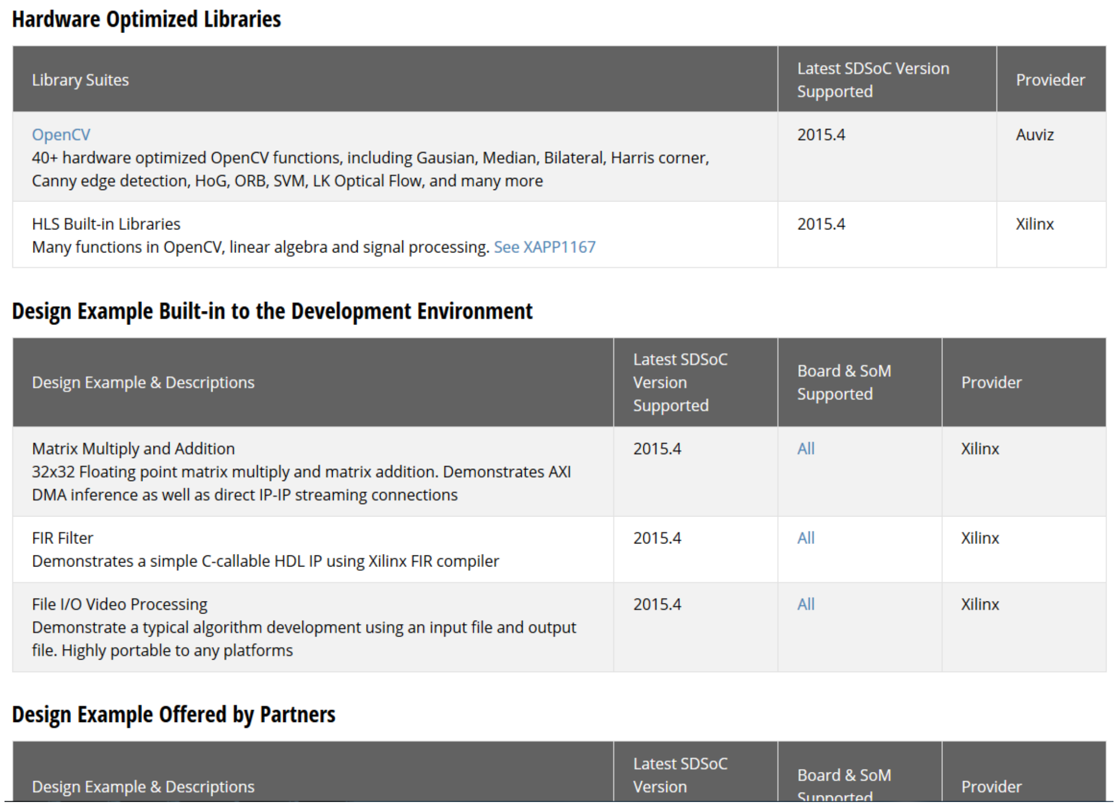
User Guides

- UG1028
  - Getting started
- UG1027
  - SDSoC flows, features & functions
- UG1146
  - Create a custom SDSoC platform and a C-callable RTL IP Library

Libraries & Design Examples

Libraries & Design Examples

- **Optimized libraries for faster programming**
- **Available from Xilinx and ecosystem partners**



Hardware Optimized Libraries

Library Suites	Latest SDSoC Version Supported	Provieder
OpenCV 40+ hardware optimized OpenCV functions, including Gausian, Median, Bilateral, Harris corner, Canny edge detection, HoG, ORB, SVM, LK Optical Flow, and many more	2015.4	Auviz
HLS Built-in Libraries Many functions in OpenCV, linear algebra and signal processing. See XAPP1167	2015.4	Xilinx

Design Example Built-in to the Development Environment

Design Example & Descriptions	Latest SDSoC Version Supported	Board & SoM Supported	Provider
Matrix Multiply and Addition 32x32 Floating point matrix multiply and matrix addition. Demonstrates AXI DMA inference as well as direct IP-IP streaming connections	2015.4	All	Xilinx
FIR Filter Demonstrates a simple C-callable HDL IP using Xilinx FIR compiler	2015.4	All	Xilinx
File I/O Video Processing Demonstrate a typical algorithm development using an input file and output file. Highly portable to any platforms	2015.4	All	Xilinx

Design Example Offered by Partners

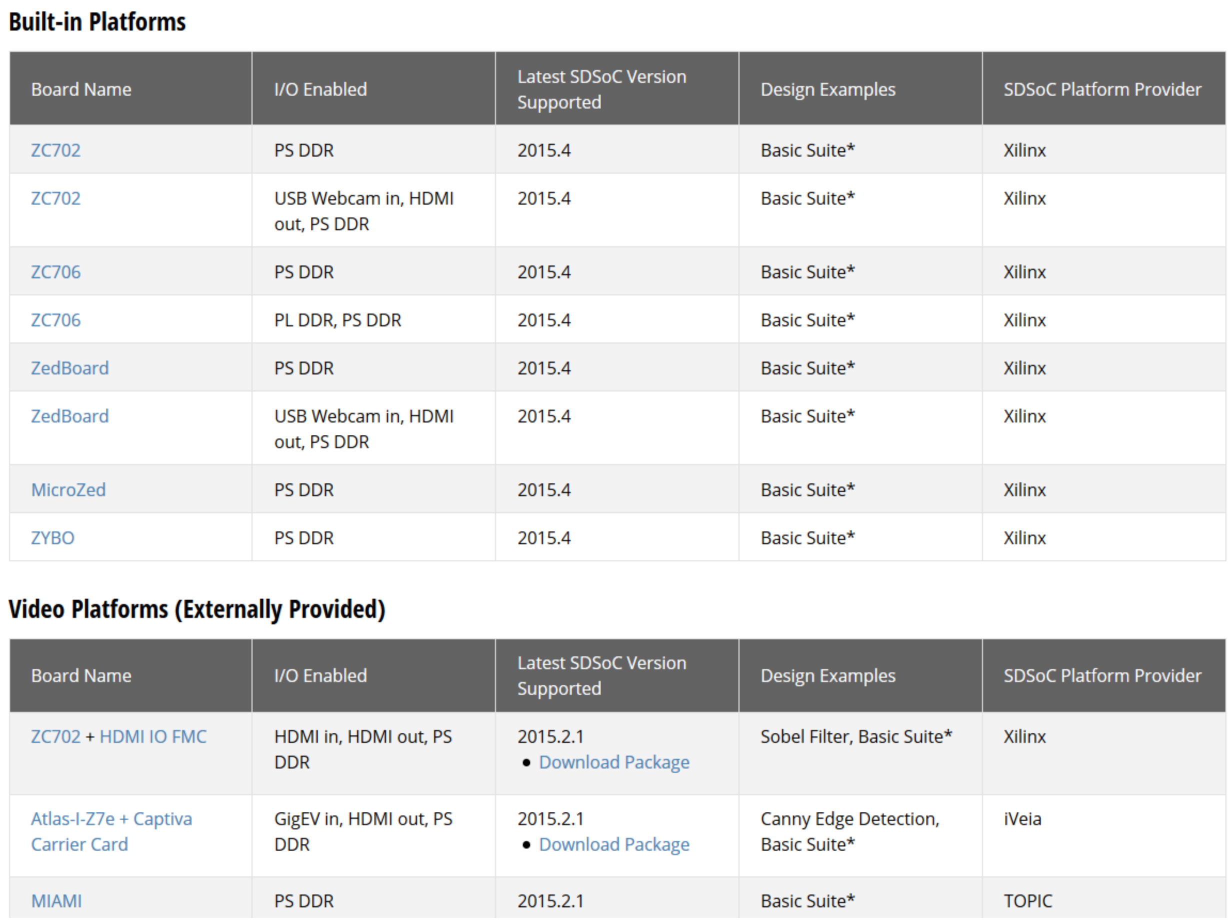
Design Example & Descriptions	Latest SDSoC Version	Board & SoM Supported	Provider

XILINX ALL PROGRAMMABLE.

Boards, Kits & Modules

Boards, Kits, & Modules

- **System-level solutions for multiple functions including video, radio & control**
- **Available from Xilinx and ecosystem partners**



Built-in Platforms

Board Name	I/O Enabled	Latest SDSoC Version Supported	Design Examples	SDSoC Platform Provider
ZC702	PS DDR	2015.4	Basic Suite*	Xilinx
ZC702	USB Webcam in, HDMI out, PS DDR	2015.4	Basic Suite*	Xilinx
ZC706	PS DDR	2015.4	Basic Suite*	Xilinx
ZC706	PL DDR, PS DDR	2015.4	Basic Suite*	Xilinx
ZedBoard	PS DDR	2015.4	Basic Suite*	Xilinx
ZedBoard	USB Webcam in, HDMI out, PS DDR	2015.4	Basic Suite*	Xilinx
MicroZed	PS DDR	2015.4	Basic Suite*	Xilinx
ZYBO	PS DDR	2015.4	Basic Suite*	Xilinx

Video Platforms (Externally Provided)

Board Name	I/O Enabled	Latest SDSoC Version Supported	Design Examples	SDSoC Platform Provider
ZC702 + HDMI IO FMC	HDMI in, HDMI out, PS DDR	2015.2.1 • Download Package	Sobel Filter, Basic Suite*	Xilinx
Atlas-I-Z7e + Captiva Carrier Card	GigEV in, HDMI out, PS DDR	2015.2.1 • Download Package	Canny Edge Detection, Basic Suite*	iVeia
MIAMI	PS DDR	2015.2.1	Basic Suite*	TOPIC

XILINX ALL PROGRAMMABLE.

Backup

SDSoC Development Environment

Thank You