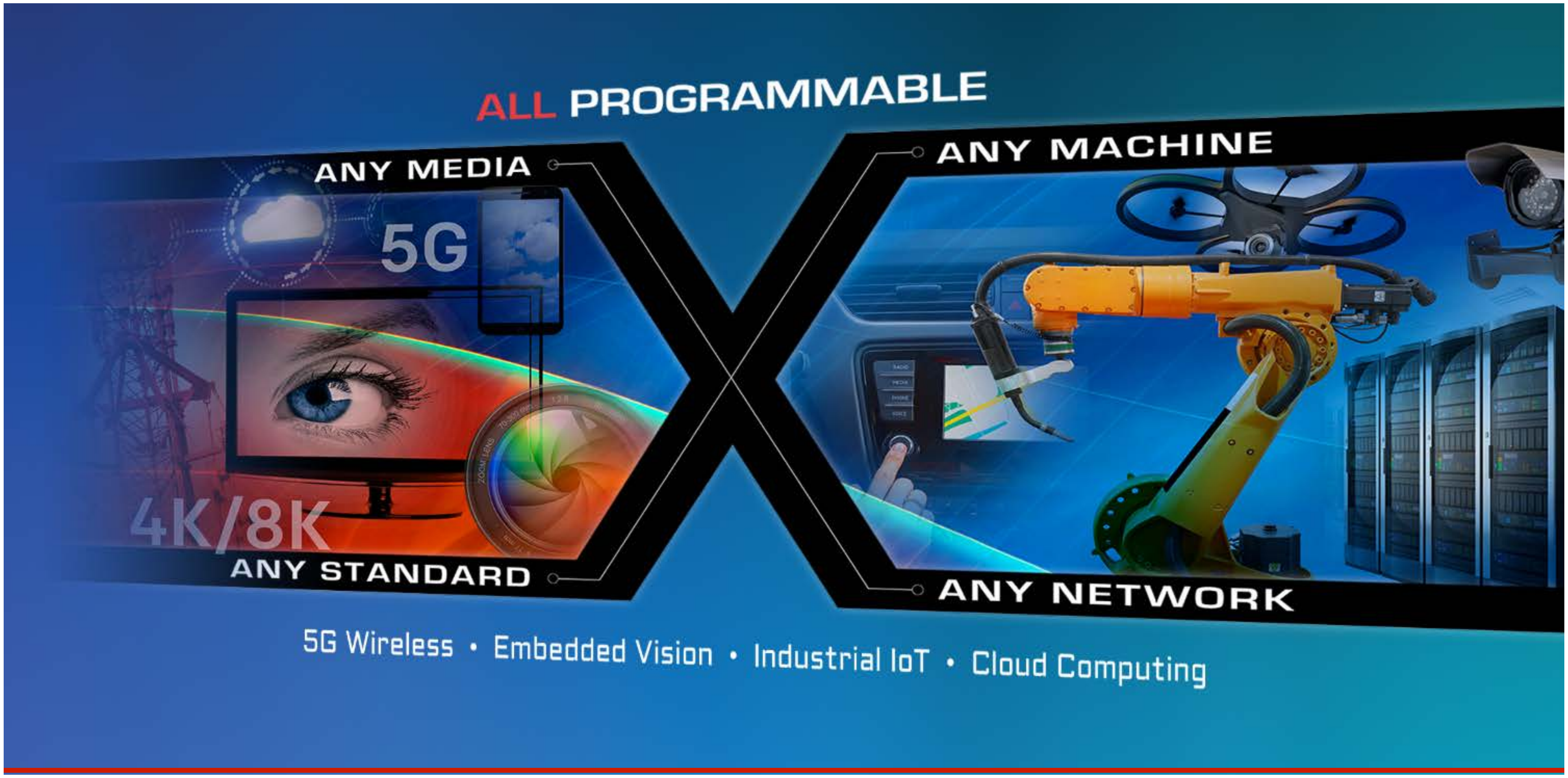


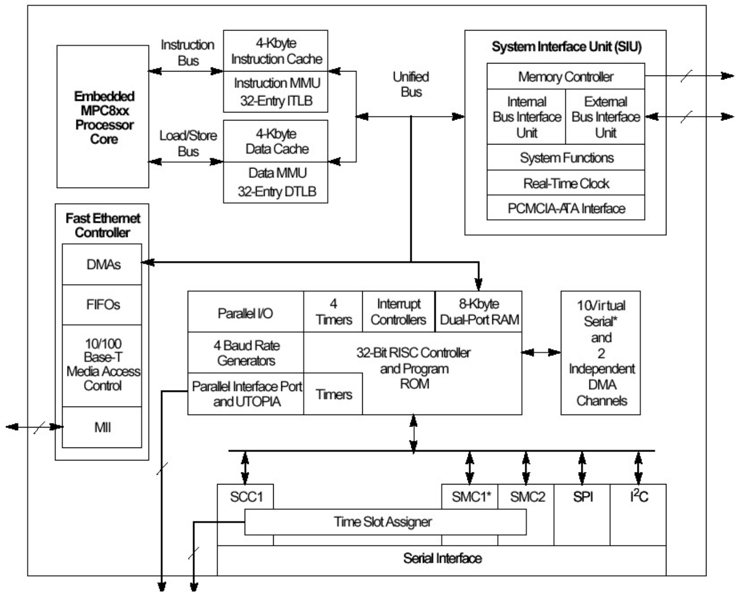
XILINX
ALL PROGRAMMABLE™

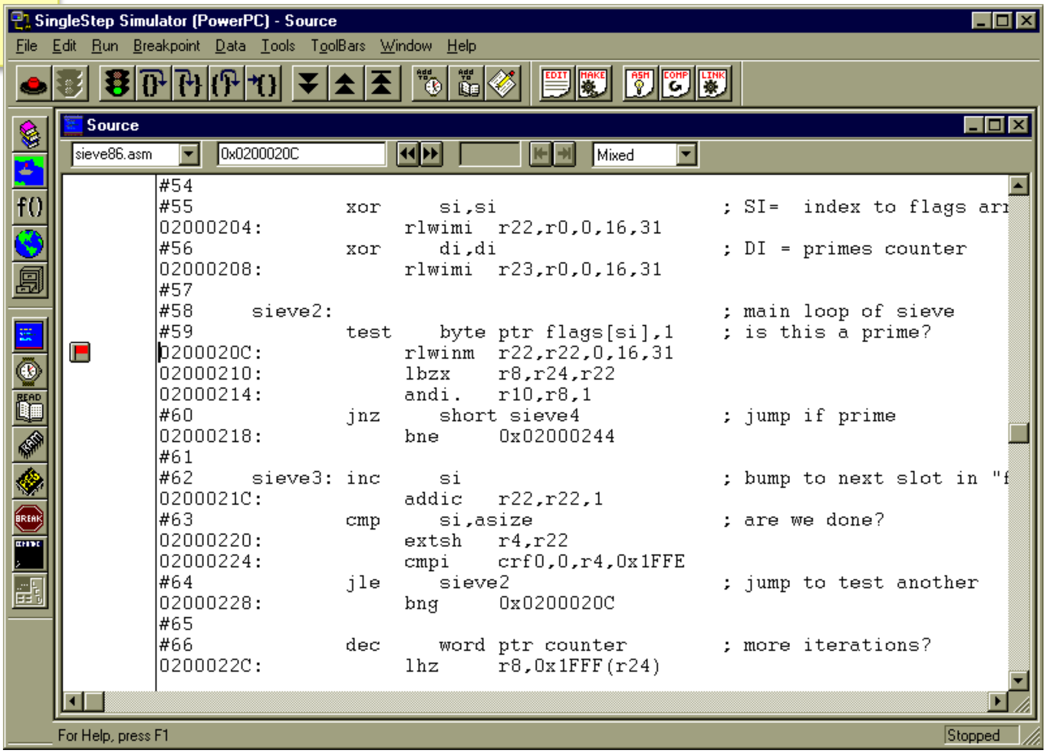
A Software Developer's Journey into a Deeply Heterogeneous World

Tomas Evensen, CTO Embedded Software, Xilinx

Embedded Development: Then

- Simple single CPU
- Most code developed internally
  - 10's of thousands of lines of code in C and assembly
- Single Real-time Operating System
- JTAG/BDM debugger
- Simple I/O





SingleStep Simulator (PowerPC) - Source
File Edit Run Breakpoint Data Tools ToolBars Window Help
Source
sieve86.asm
0x0200020C
Mixed

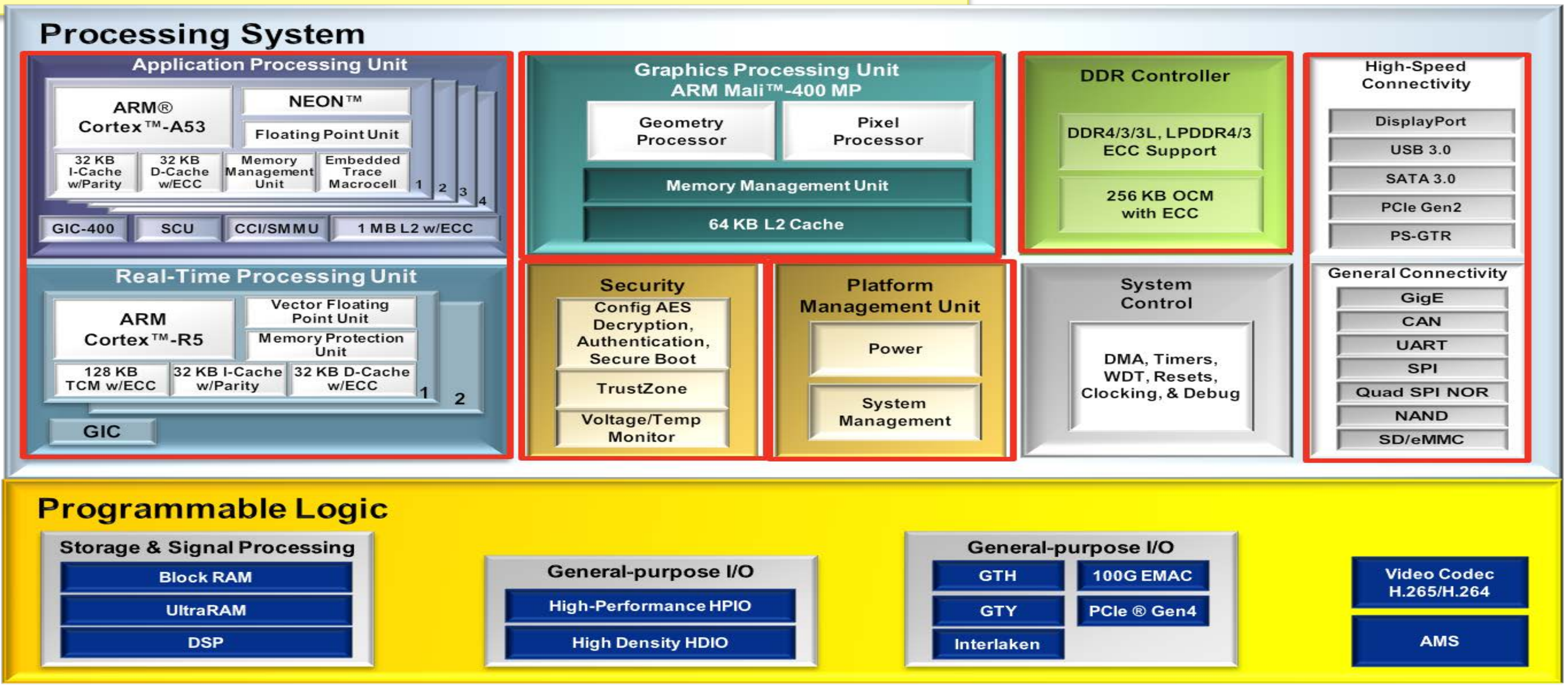
```
#54
#55                xor     si,si                     ; SI=  index to flags arr
02000204:                  rlwimi  r22,r0,0,16,31
#56                xor     di,di                     ; DI = primes counter
02000208:                  rlwimi  r23,r0,0,16,31
#57
#58      sieve2:                                     ; main loop of sieve
#59                test    byte ptr flags[si],1      ; is this a prime?
0200020C:                  rlwinm  r22,r22,0,16,31
02000210:                  lbzx    r8,r24,r22
02000214:                  andi.   r10,r8,1
#60                jnz     short sieve4              ; jump if prime
02000218:                  bne     0x02000244
#61
#62      sieve3:   inc     si                        ; bump to next slot in "f
0200021C:                  addic   r22,r22,1
#63                cmp     si,asize                  ; are we done?
02000220:                  extsh   r4,r22
02000224:                  cmpi    crf0,0,r4,0x1FFE
#64                jle     sieve2                    ; jump to test another
02000228:                  bng     0x0200020C
#65
#66                dec     word ptr counter          ; more iterations?
0200022C:                  lhz     r8,0x1FFF(r24)
```

For Help, press F1 — Stopped

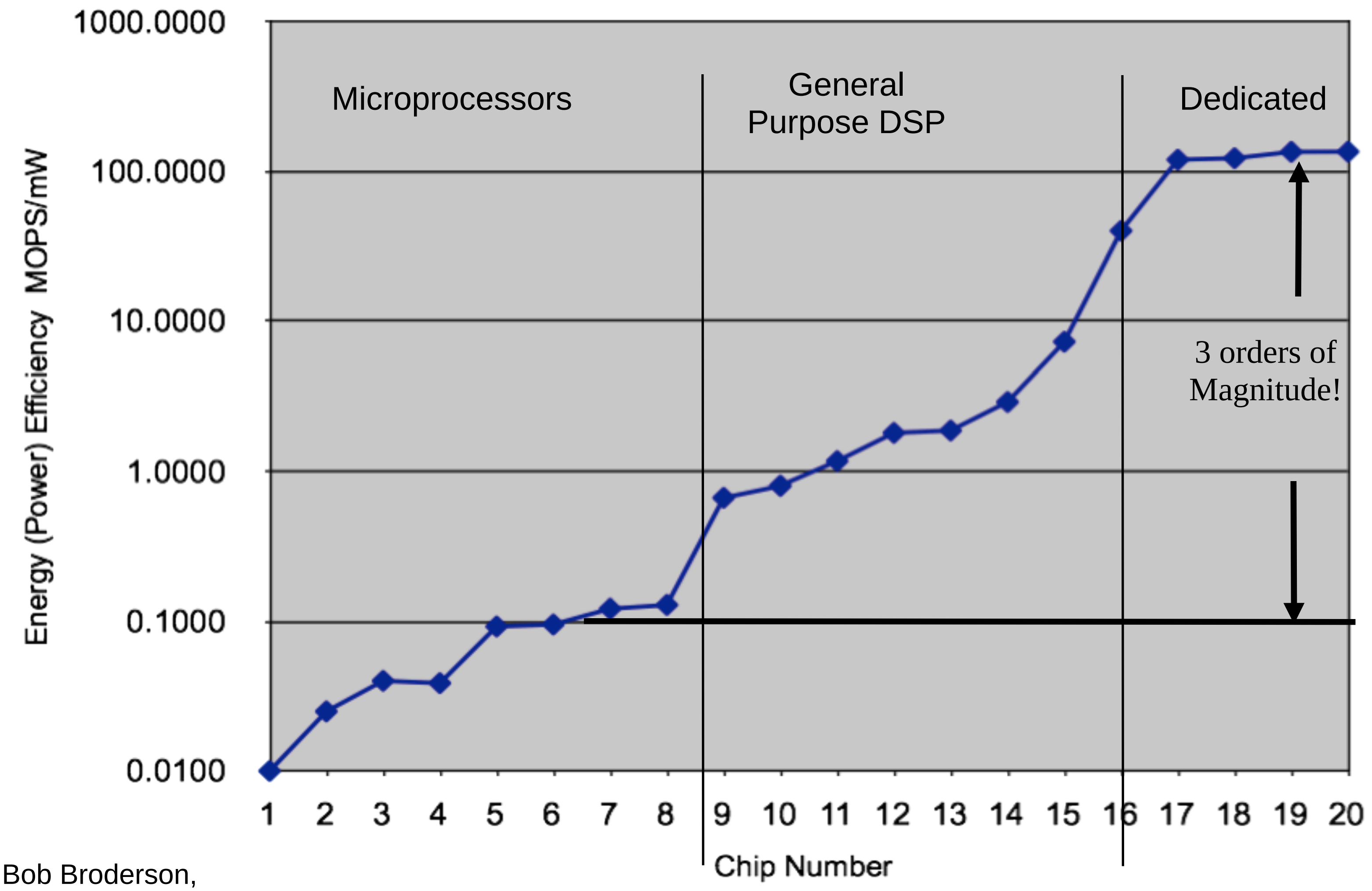
Embedded Development: Now

- Multiple heterogeneous CPUs
- Multiple accelerators and programmable logic
- Millions of lines of code - Mostly from other places like open source
- Multiple Operating Systems (i.e. Linux + RTOS)
- JTAG debugger
- Safety and Security concerns

Xilinx Zynq MPSoC

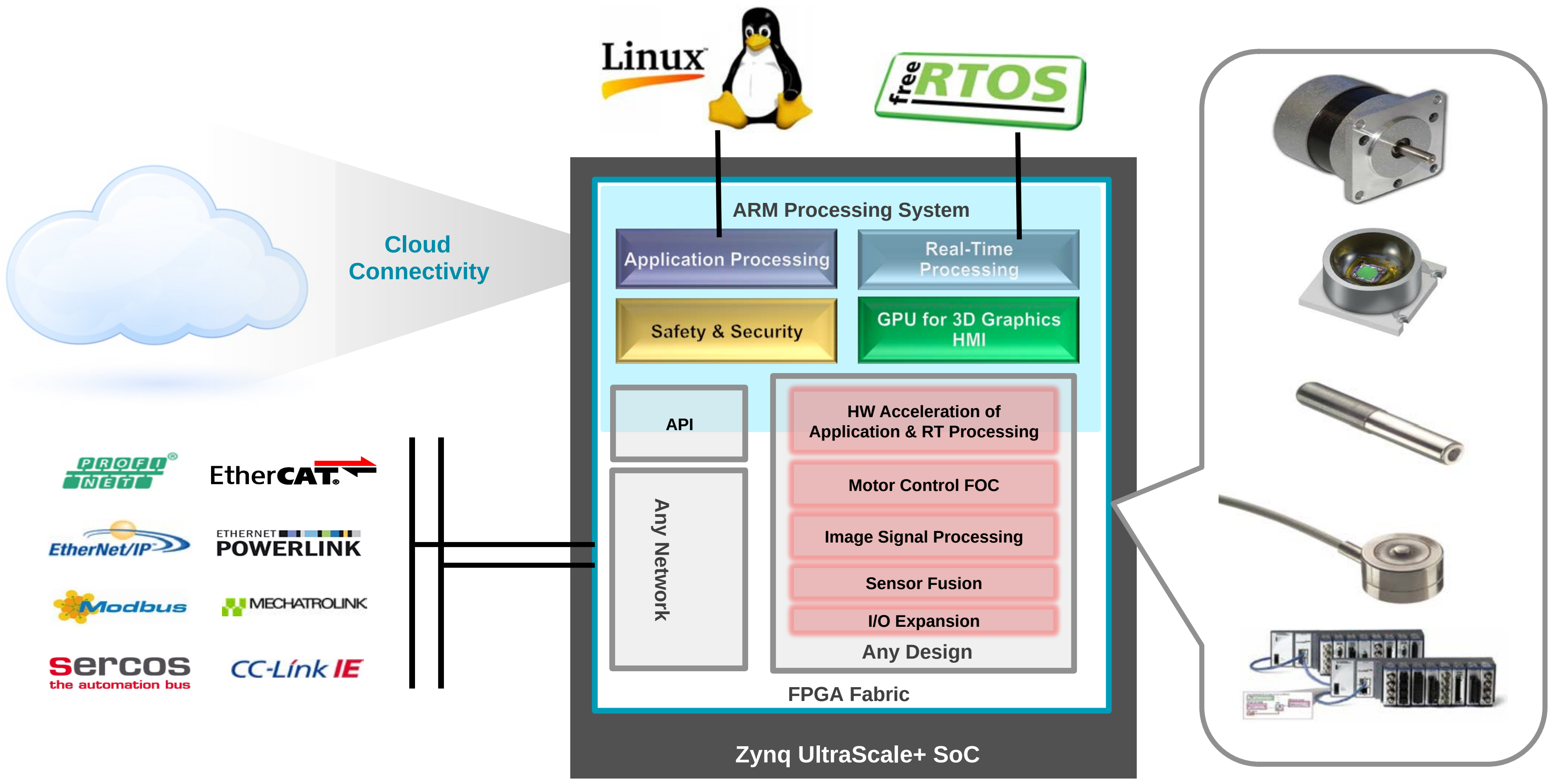


Dedicated Hardware is Energy Efficient



Courtesy Bob Broderson,
based on published results at ISSCC conferences.

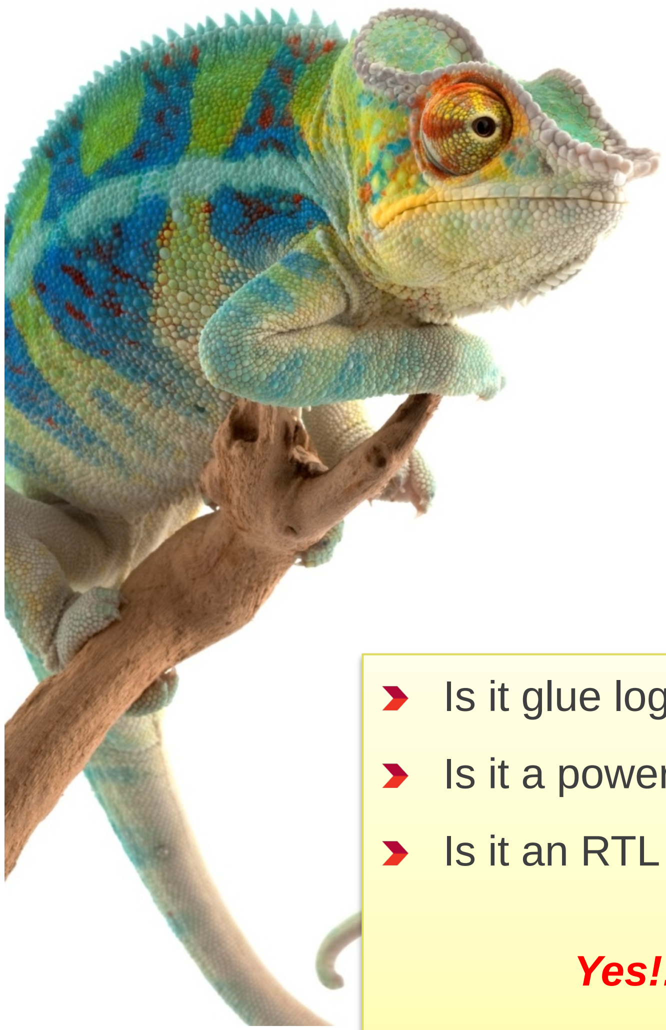
Heterogeneous Example: IIoT Gateway

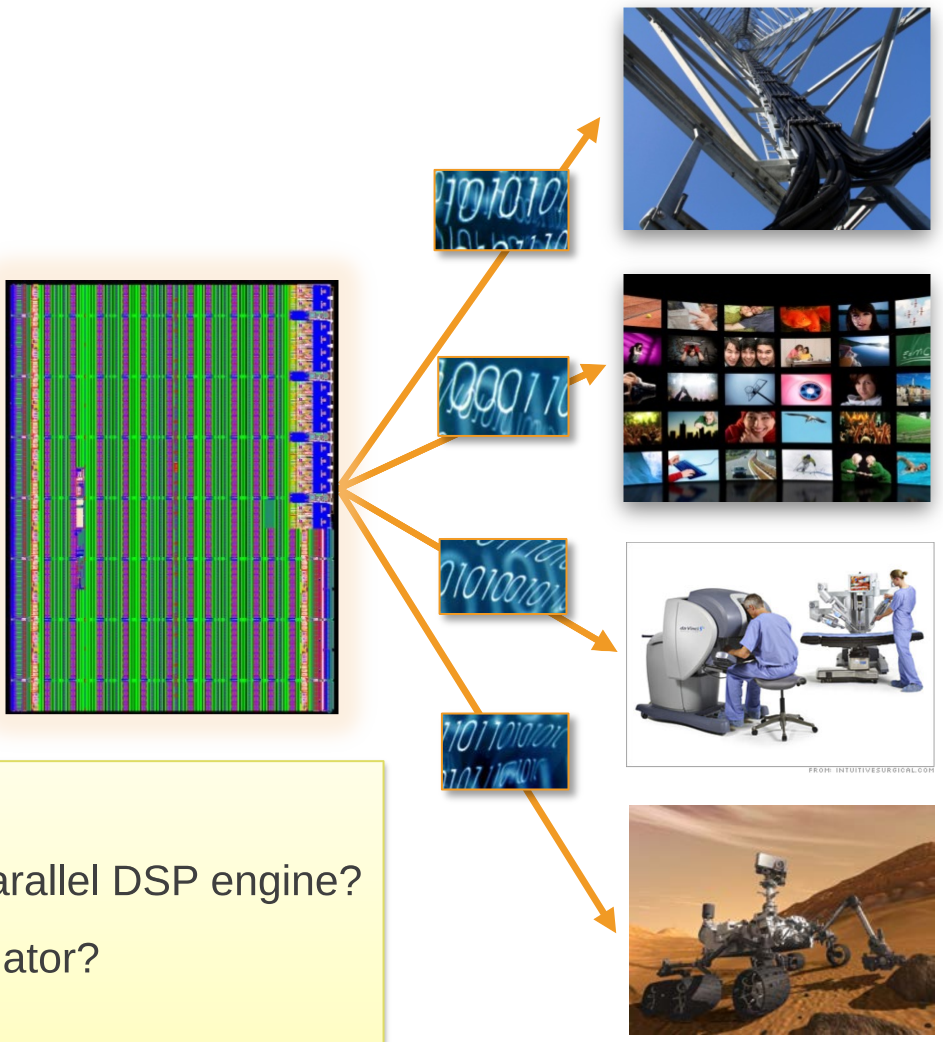


Expertise Needed All the Way from a System Level to Cloud Connectivity

XILINX ALL PROGRAMMABLE.

FPGA – The “Chameleon” Chip





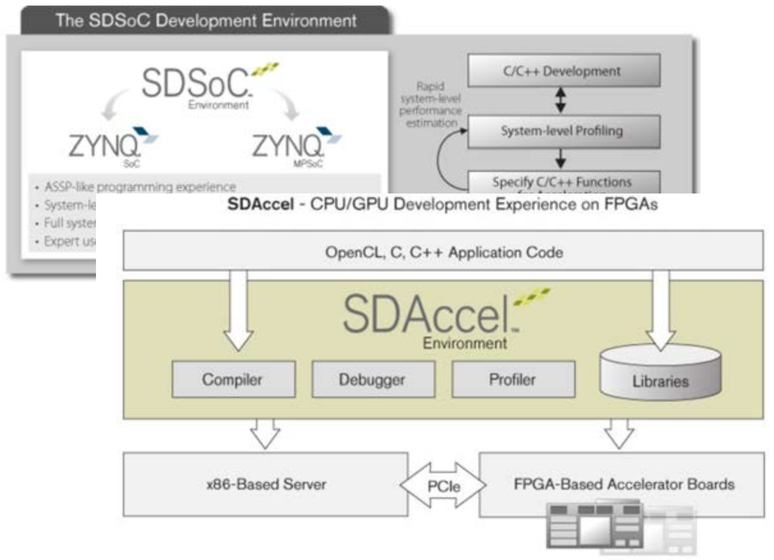
- Is it glue logic?
- Is it a powerful parallel DSP engine?
- Is it an RTL simulator?

Yes!!!

And more…

FPGA – Reaching New Developers

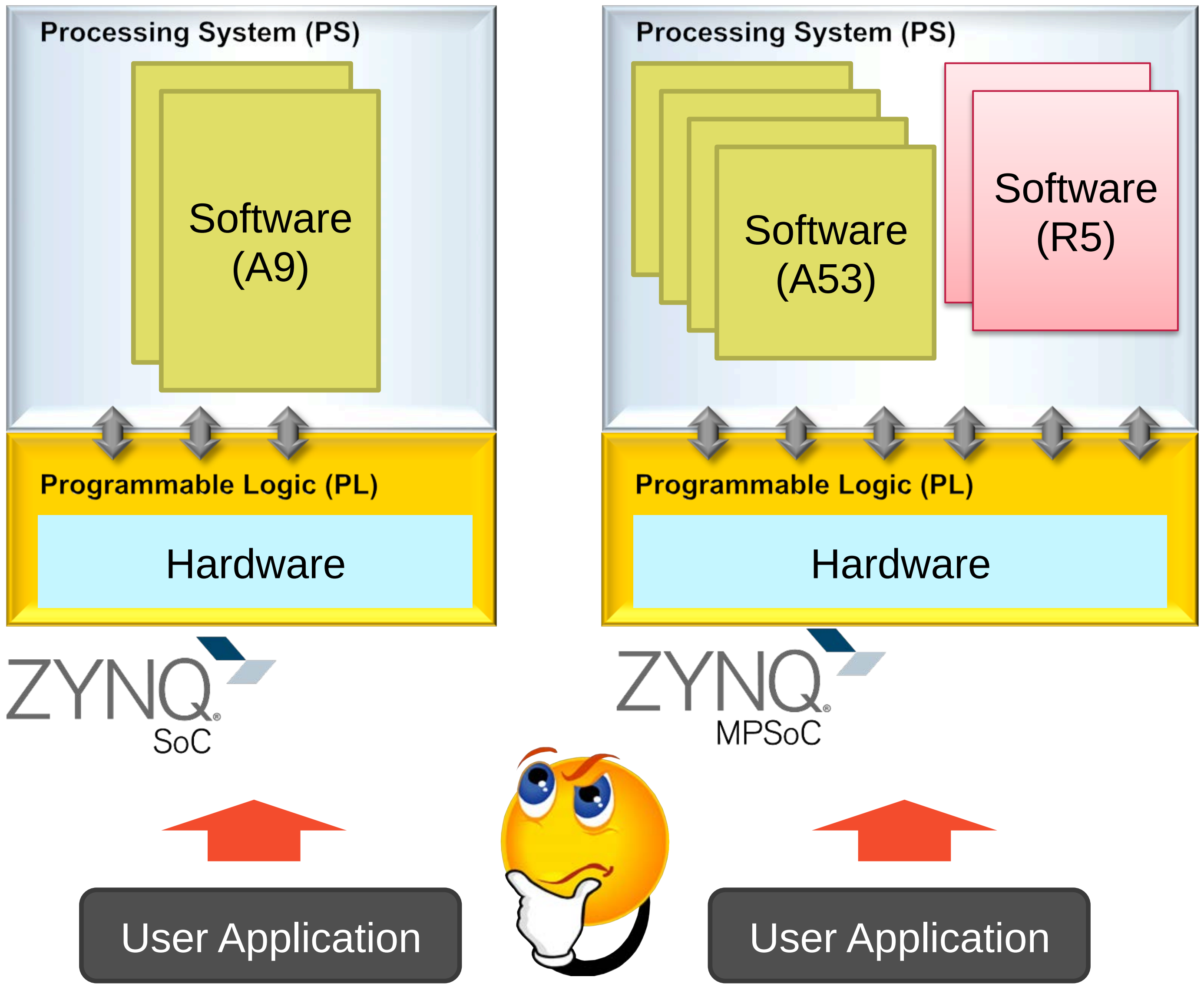
- **Limited pool of FPGA developers**
  - – Need to reach software developers
  - – Software developers are different!
- **Key to reach software developers**
  1. Create libraries so they can utilize accelerators written by others
  2. Create tools so they can utilize FPGA without RTL


The SDSoC Development Environment
SDSoC Environment
ZYNQ SoC
ZYNQ MPSoC
• ASSP-like programming experience
• System-le
• Full syste
• Expert us
Rapid system-level performance estimation
C/C++ Development
System-level Profiling
Specify C/C++ Functions
SDAccel - CPU/GPU Development Experience on FPGAs
OpenCL, C, C++ Application Code
SDAccel Environment
Compiler
Debugger
Profiler
Libraries
x86-Based Server
PCIe
FPGA-Based Accelerator Boards

XILINX ALL PROGRAMMABLE.

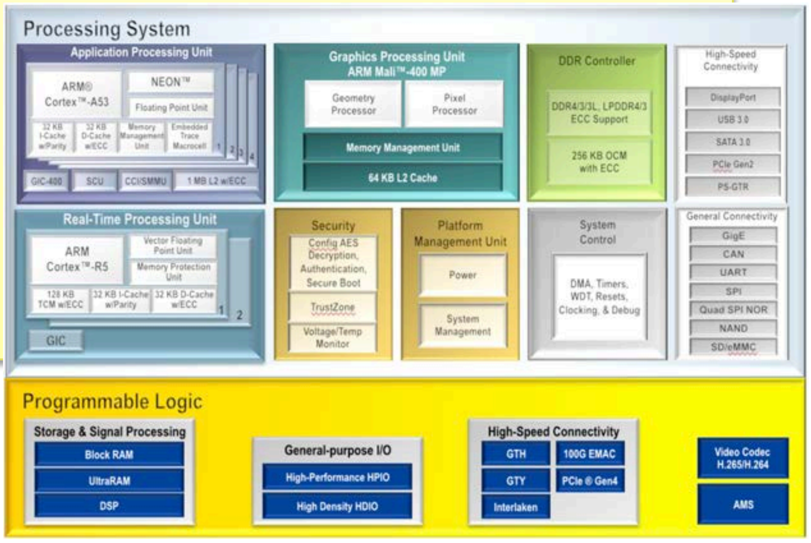
Heterogeneous Software Development

Mapping Applications to Heterogeneous Systems



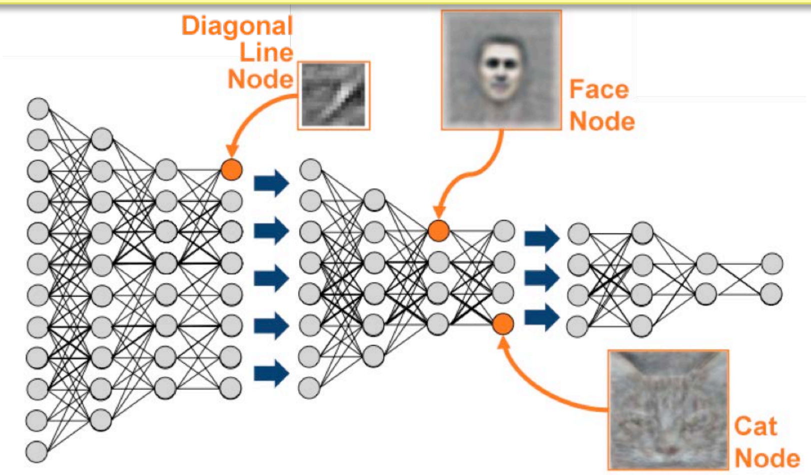
Components for Heterogeneous SW Development

- **Accelerated libraries and frameworks for common functions**
  - E.g. OpenCV, CNN, …
- **Support for multiple types of Operating Environments**
  - Solid Linux support, bare metal, FreeRTOS, 3rd party RTOS, Windows EC
  - Mixing of OS's through AMP and hypervisors
- **System debugger – Unifying debug/profile**
  - Debug across cores and FPGA including profiling and trace
- **FPGA Compiler – SDSoC**
  - Write code for FPGA using C/C++/OpenCL
  - Automate the "glue" between execution engines
- **Other**
  - Virtual Prototyping for complete system



Framework Programming: Deep Learning

- **Many embedded problems are being converted to use deep learning**
  - Embedded vision, speech, …
  - Using neural networks of different kinds, e.g. CNN, …
- **Neural networks are “programmed” through learning**
- **Neural networks are typically controlled by frameworks**
  - Caffe, Tensorflow, Torch, Theano, …
- **Neural networks are very computation intensive**
- **FPGAs can be very efficient for neural networks**
  - Combination of fixed point, flexible routing, memory hierarchies and DSPs
  - By supporting existing framework, programmers can avoid RTL

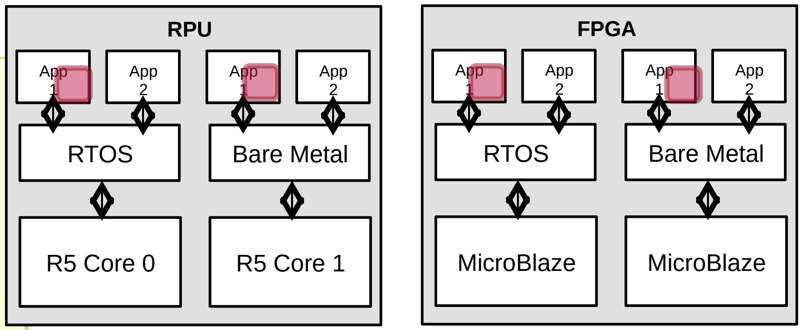


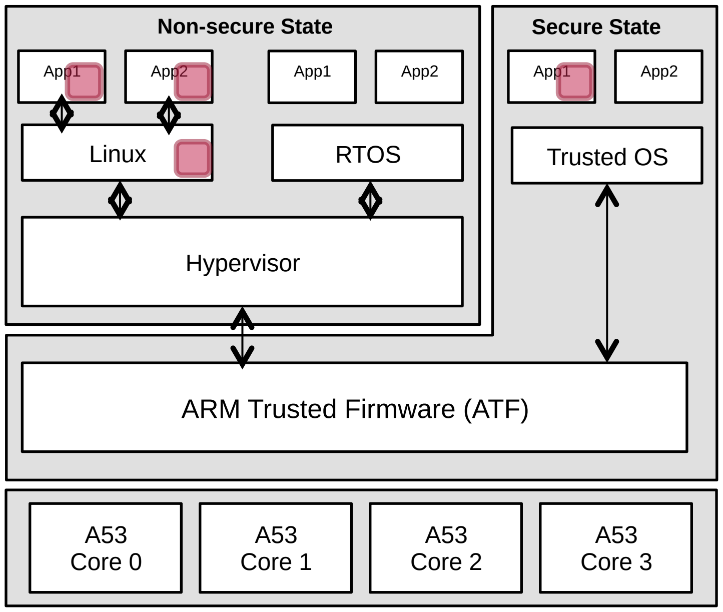
AlexNet Calculations

	Output Feature Maps			Filter Sizing			MACs
	Rows	Cols	Depth	Dim	Depth		conv
conv1	55	55	64	11	3		70,276,800
conv2	27	27	192	5	64		223,948,800
conv3	13	13	384	3	192		112,140,288
conv4	13	13	256	3	384		149,520,384
conv5	13	13	256	3	256		99,680,256
fc6	6	6	256		4096		37,748,736
fc7	1	1	4096		4096		16,777,216
fc8	1	1	4096		1000		4,096,000
						Total	**714,188,480**

OpenAMP: A Standard for Multi-OS Systems

- What is OpenAMP?
  - A standard for mixing embedded Operating Systems
  - An Open Source project
- Trend to combine Operating Systems
  - Linux is used in majority of use cases
  - Many free and commercial RTOS's are being used
  - Bare metal (no OS) is common on smaller cores
- Why multiple Operating Systems?
  - Heterogeneous cores
  - Different needs
    - Real-time vs. general purpose
    - Different Safety/Security levels
    - Legacy
    - GPL avoidance
- Safety and Security issues common
  - Affects boot order, messaging implementation, …

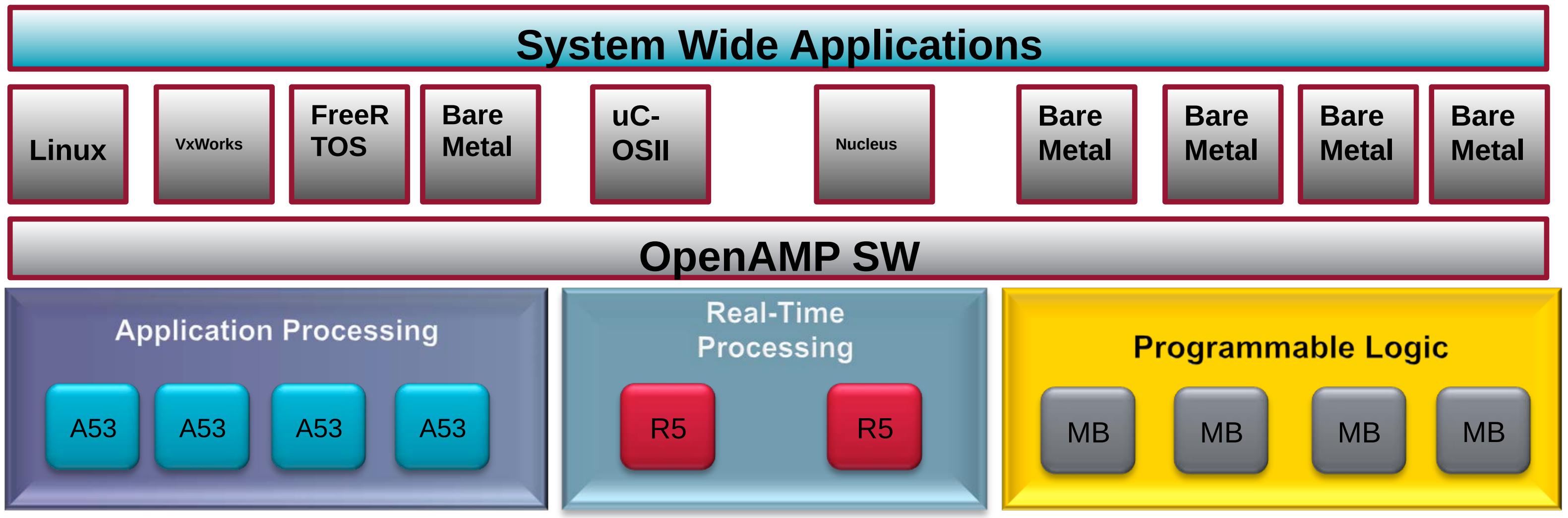




🟥 - **Examples of OpenAMP applications**

XILINX ALL PROGRAMMABLE.

OpenAMP Capabilities



- **Provides a Layer for Applications**
  - Standard API's that allow applications to be ported across processors and operating systems
- **System Development**
  - Provides a wide rage of capabilities needed to deploy applications across asymmetric computing elements

- **Inter-OS & Inter Processor Communication**
  - Send messages back and forth
- **OS Management**
  - Provides booting/rebooting of processors
- **Two Implementations**
  - GPL implementation in Linux kernel
  - BSD implementation for RTOS/BM/Linux user space

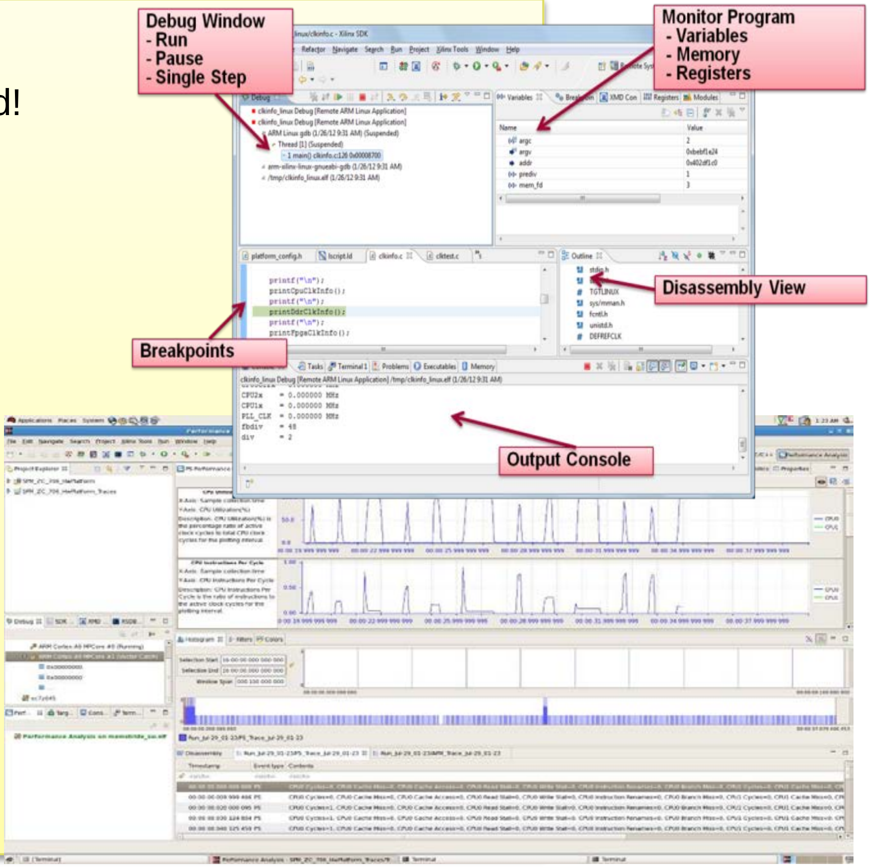
Software Development Tools (SDK)

2015 UBM Electronics Embedded Markets Study

What are the most important factors in choosing a processor?

71% - Software Development tools

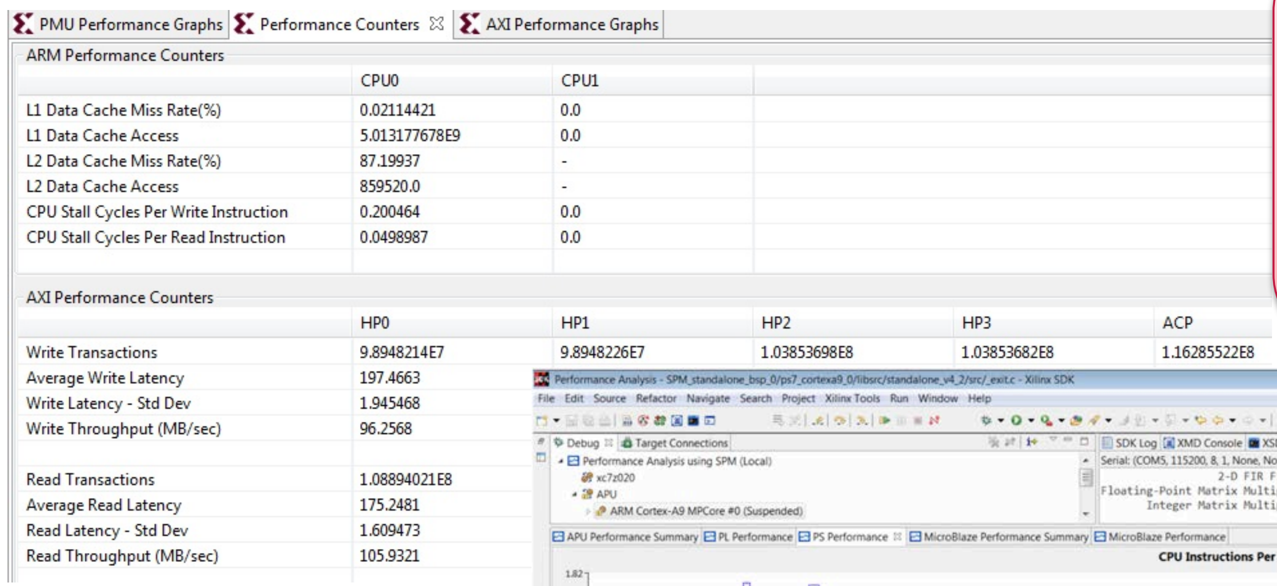
- **Complete system visibility needed**
  - Heterogeneous debugging and analysis is *very* hard!
  - Especially timing related problems
- **Tools Features:**
  - Heterogeneous system Level Debugging
    - Visibility into both CPUs and FPGA
  - Integrated performance profiling
    - Which parts of the chip are busy?
    - Measure processor and bus activities
    - Integrated traffic generator
  - System event trace
    - What is happening in the chip over time?
    - Combined time line for SW and HW events
  - Based on standards – Open source Eclipse, TCF



Strong system level tools are critical for heterogeneous development

XILINX ALL PROGRAMMABLE.

Performance Data

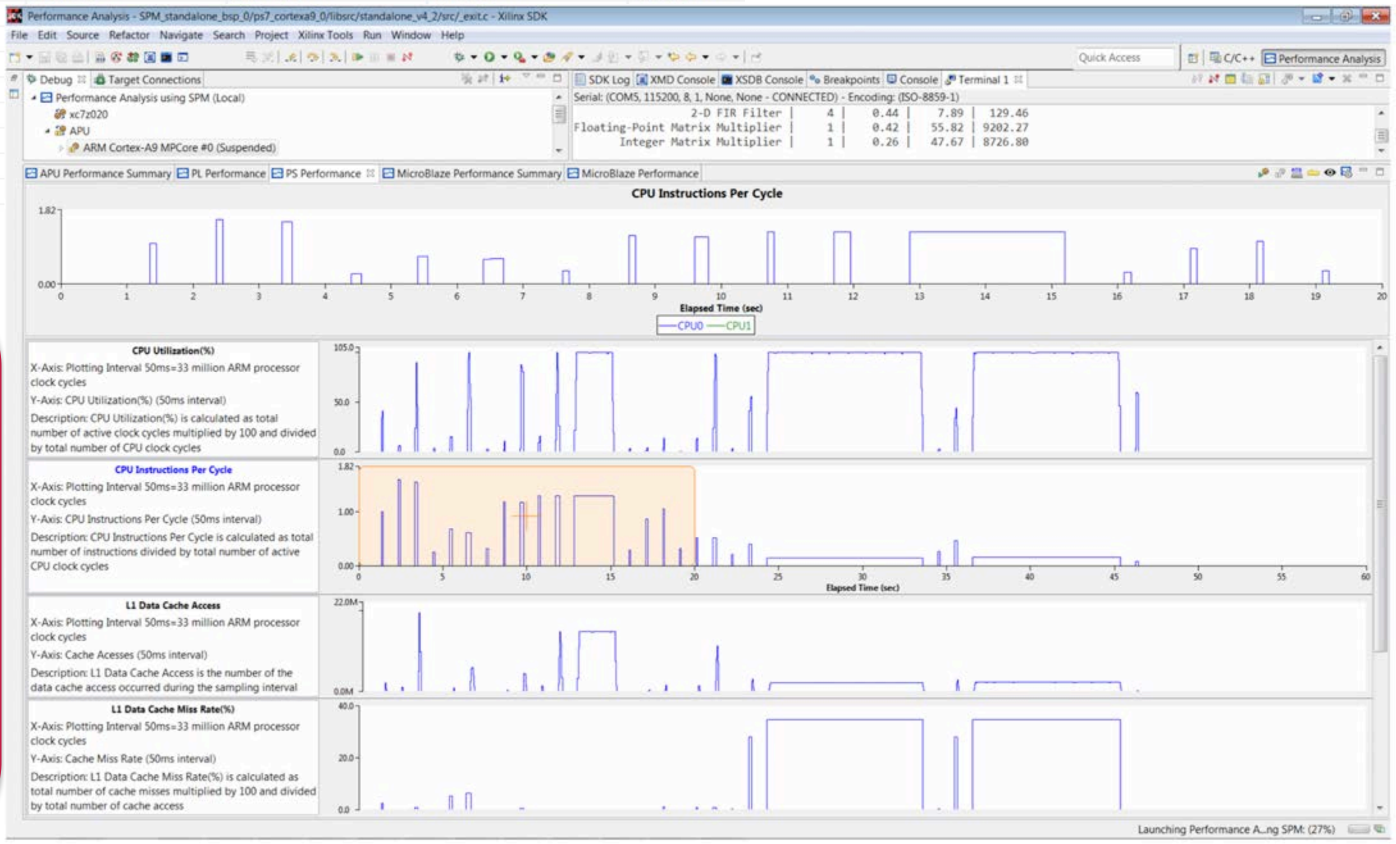


ARM Performance Counters

	CPU0	CPU1
L1 Data Cache Miss Rate(%)	0.02114421	0.0
L1 Data Cache Access	5.013177678E9	0.0
L2 Data Cache Miss Rate(%)	87.19937	-
L2 Data Cache Access	859520.0	-
CPU Stall Cycles Per Write Instruction	0.200464	0.0
CPU Stall Cycles Per Read Instruction	0.0498987	0.0

AXI Performance Counters

	HP0	HP1	HP2	HP3	ACP
Write Transactions	9.8948214E7	9.8948226E7	1.03853698E8	1.03853682E8	1.16285522E8
Average Write Latency	197.4663				
Write Latency - Std Dev	1.945468				
Write Throughput (MB/sec)	96.2568				
Read Transactions	1.08894021E8				
Average Read Latency	175.2481				
Read Latency - Std Dev	1.609473				
Read Throughput (MB/sec)	105.9321				


Performance Analysis - SPM_standalone_bsp_0/ps7_cortexa9_0/libsrc/standalone_v4_2/src/_exit.c - Xilinx SDK
File Edit Source Refactor Navigate Search Project Xilinx Tools Run Window Help
Quick Access
C/C++
Performance Analysis
Debug
Target Connections
Performance Analysis using SPM (Local)
xc7z020
APU
ARM Cortex-A9 MPCore #0 (Suspended)
SDK Log
XMD Console
XSDB Console
Breakpoints
Console
Terminal 1
APU Performance Summary
PL Performance
PS Performance
MicroBlaze Performance Summary
MicroBlaze Performance
CPU Instructions Per Cycle
Elapsed Time (sec)
CPU0
CPU1
Launching Performance A...ng SPM: (27%)

```
Serial: (COM5, 115200, 8, 1, None, None - CONNECTED) - Encoding: (ISO-8859-1)
                  2-D FIR Filter |      4 |     0.44 |     7.89 |  129.46
Floating-Point Matrix Multiplier |      1 |     0.42 |    55.82 | 9202.27
       Integer Matrix Multiplier |      1 |     0.26 |    47.67 | 8726.80
```

CPU Utilization(%)
X-Axis: Plotting Interval 50ms=33 million ARM processor clock cycles
Y-Axis: CPU Utilization(%) (50ms interval)
Description: CPU Utilization(%) is calculated as total number of active clock cycles multiplied by 100 and divided by total number of CPU clock cycles

CPU Instructions Per Cycle
X-Axis: Plotting Interval 50ms=33 million ARM processor clock cycles
Y-Axis: CPU Instructions Per Cycle (50ms interval)
Description: CPU Instructions Per Cycle is calculated as total number of instructions divided by total number of active CPU clock cycles

L1 Data Cache Access
X-Axis: Plotting Interval 50ms=33 million ARM processor clock cycles
Y-Axis: Cache Acesses (50ms interval)
Description: L1 Data Cache Access is the number of the data cache access occurred during the sampling interval

L1 Data Cache Miss Rate(%)
X-Axis: Plotting Interval 50ms=33 million ARM processor clock cycles
Y-Axis: Cache Miss Rate (50ms interval)
Description: L1 Data Cache Miss Rate(%) is calculated as total number of cache misses multiplied by 100 and divided by total number of cache access

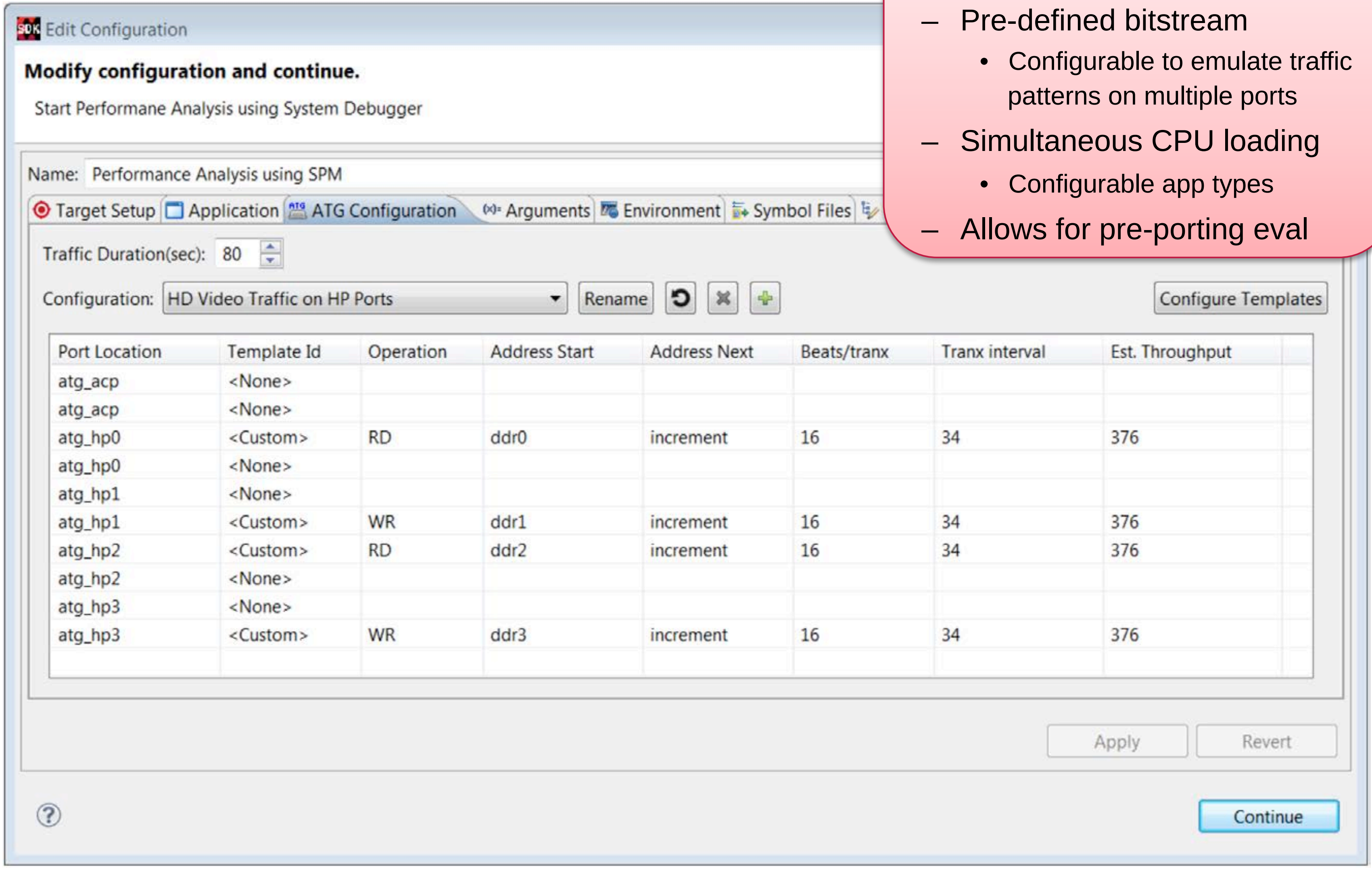
Live tables

– ARM performance registers
  • Cache misses, IPC, …
– AXI performance registers
  • Transactions, latency, …
– Non-intrusive JTAG profiling

Timeline plot

– Correlate performance
  • Cache, busses, CPU, …
– Examples:
  • How does ACP traffic affect cache miss rate?
  • How balanced are the busses?
  • How does changing mem access priority affect throughput?

Evaluate Performance - Traffic Generation



Edit Configuration

Modify configuration and continue.

Start Performane Analysis using System Debugger

Name: Performance Analysis using SPM

Target Setup | Application | ATG Configuration | Arguments | Environment | Symbol Files

Traffic Duration(sec): 80

Configuration: HD Video Traffic on HP Ports

Rename | Configure Templates

Port Location	Template Id	Operation	Address Start	Address Next	Beats/tranx	Tranx interval	Est. Throughput
atg_acp	<None>						
atg_acp	<None>						
atg_hp0	<Custom>	RD	ddr0	increment	16	34	376
atg_hp0	<None>						
atg_hp1	<None>						
atg_hp1	<Custom>	WR	ddr1	increment	16	34	376
atg_hp2	<Custom>	RD	ddr2	increment	16	34	376
atg_hp2	<None>						
atg_hp3	<None>						
atg_hp3	<Custom>	WR	ddr3	increment	16	34	376

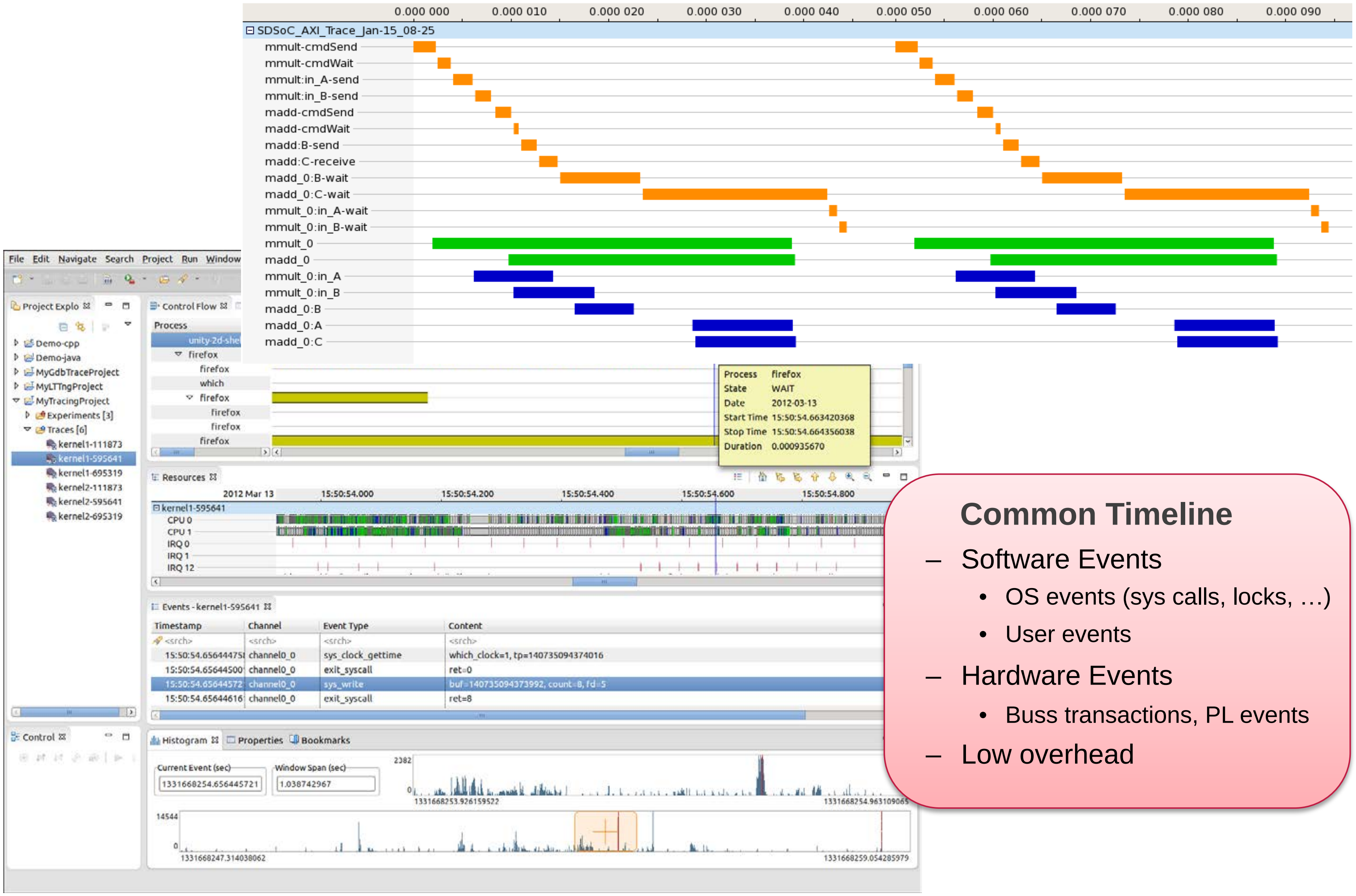
Apply | Revert | Continue

Generate Traffic Patterns

- Pre-defined bitstream
  - Configurable to emulate traffic patterns on multiple ports
- Simultaneous CPU loading
  - Configurable app types
- Allows for pre-porting eval

XILINX ALL PROGRAMMABLE.

Event Trace to Dissect Timing Issues



Process	firefox
State	WAIT
Date	2012-03-13
Start Time	15:50:54.663420368
Stop Time	15:50:54.664356038
Duration	0.000935670

Timestamp	Channel	Event Type	Content
<srch>	<srch>	<srch>	<srch>
15:50:54.65644475	channel0_0	sys_clock_gettime	which_clock=1, tp=140735094374016
15:50:54.65644500	channel0_0	exit_syscall	ret=0
15:50:54.65644572	channel0_0	sys_write	buf=140735094373992, count=8, fd=5
15:50:54.65644616	channel0_0	exit_syscall	ret=8

Current Event (sec): 1331668254.656445721

Window Span (sec): 1.038742967

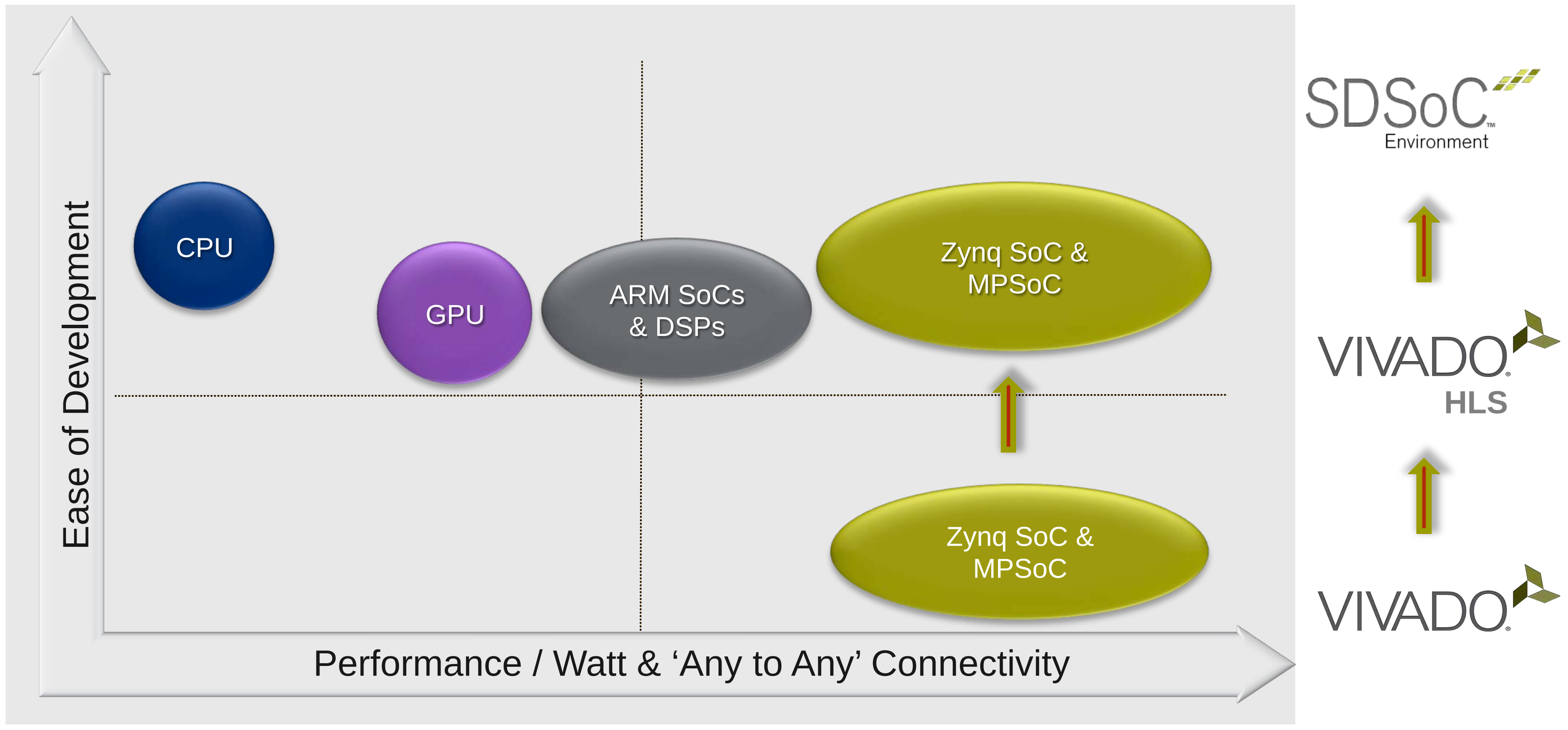
Common Timeline

- Software Events
  - OS events (sys calls, locks, …)
  - User events
- Hardware Events
  - Buss transactions, PL events
- Low overhead

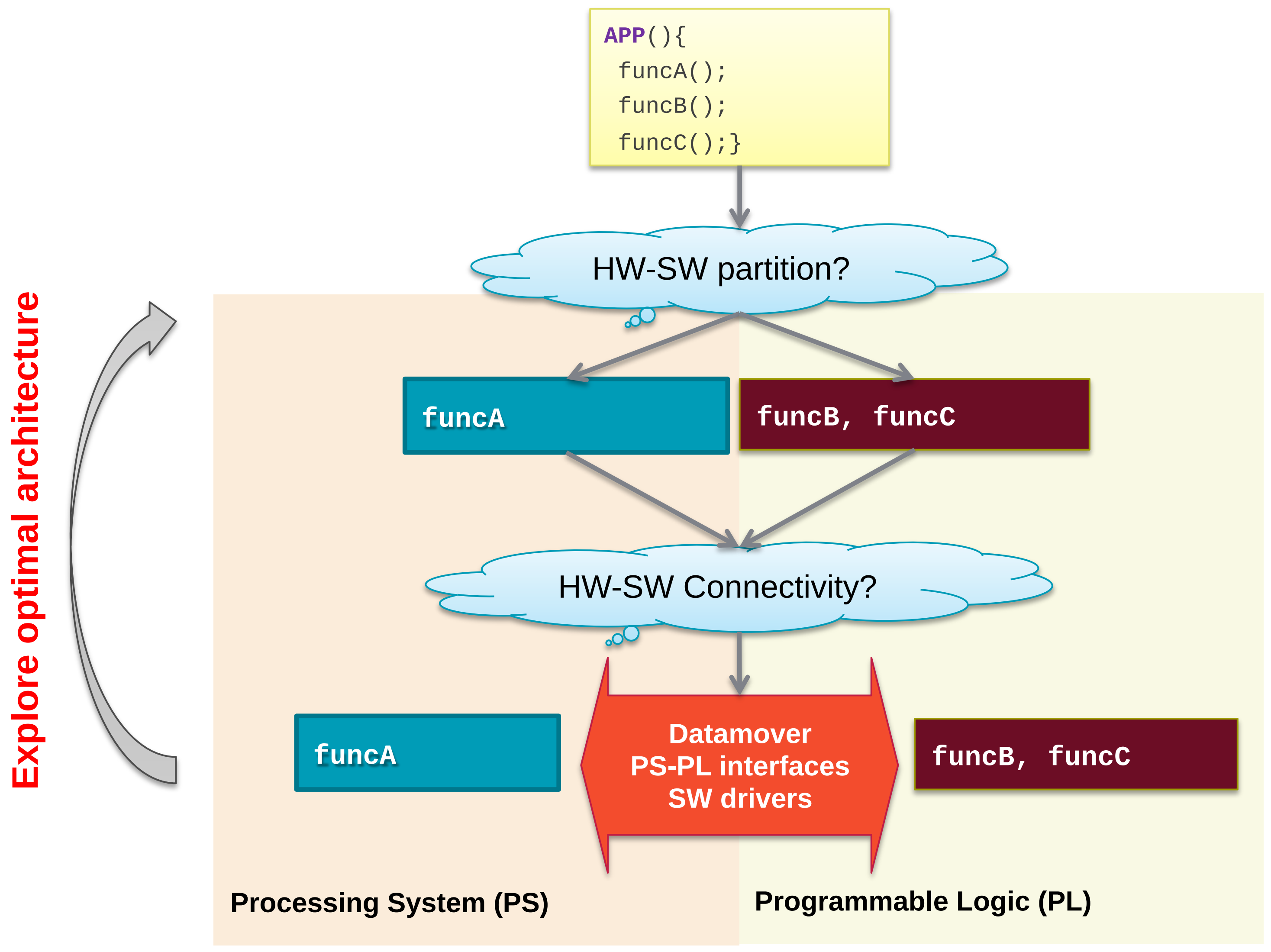
XILINX ALL PROGRAMMABLE.

SDSoC: FPGA Development through Software

FPGA Productivity with Technology Advancement

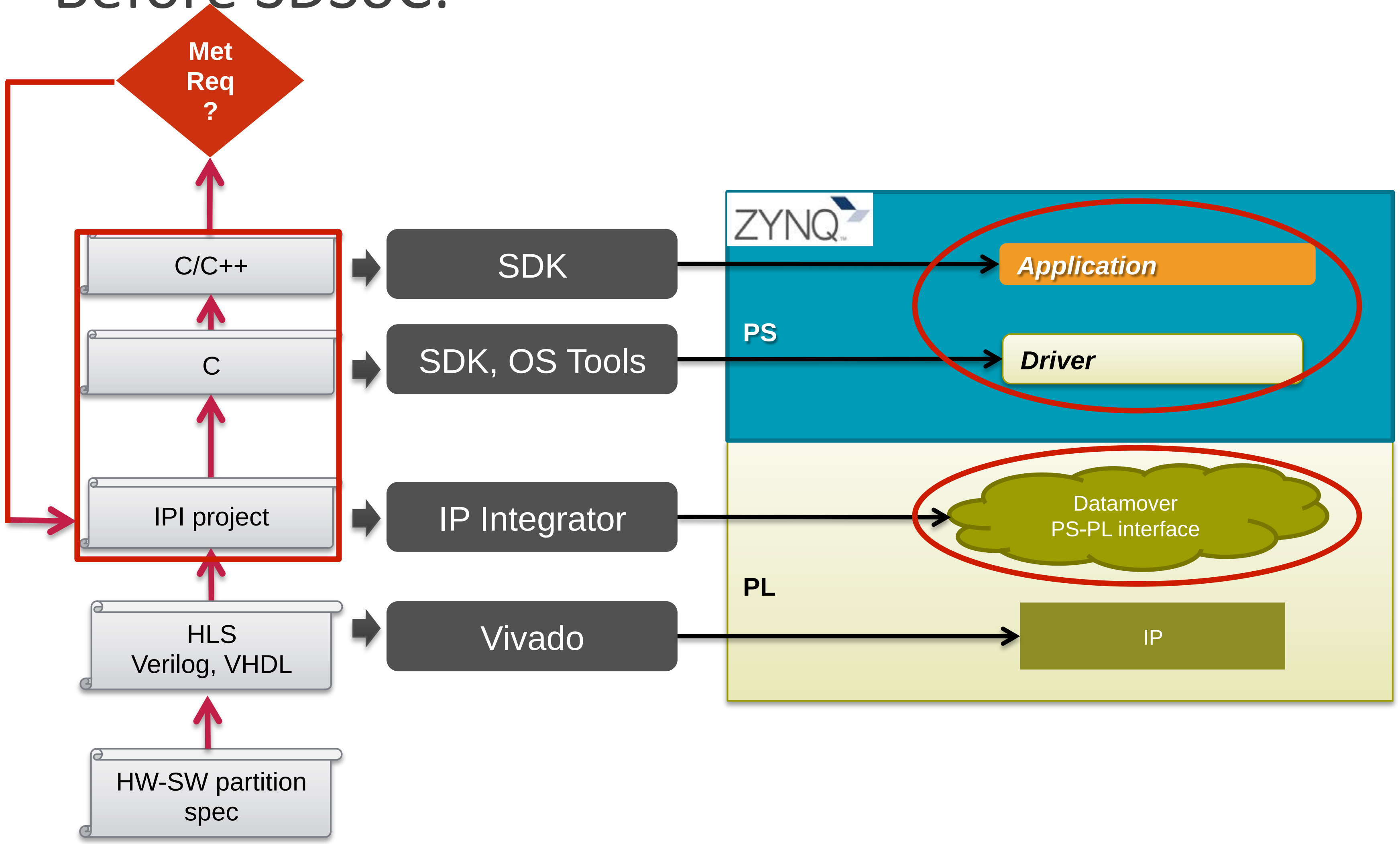


Typical Zynq Development Flow



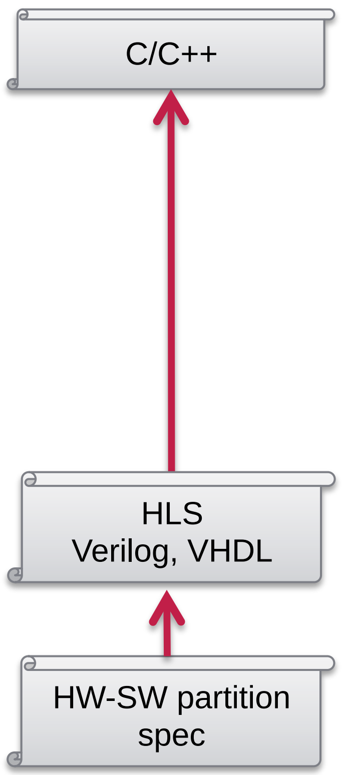
```
APP(){
 funcA();
 funcB();
 funcC();}
```

Before SDSoC:



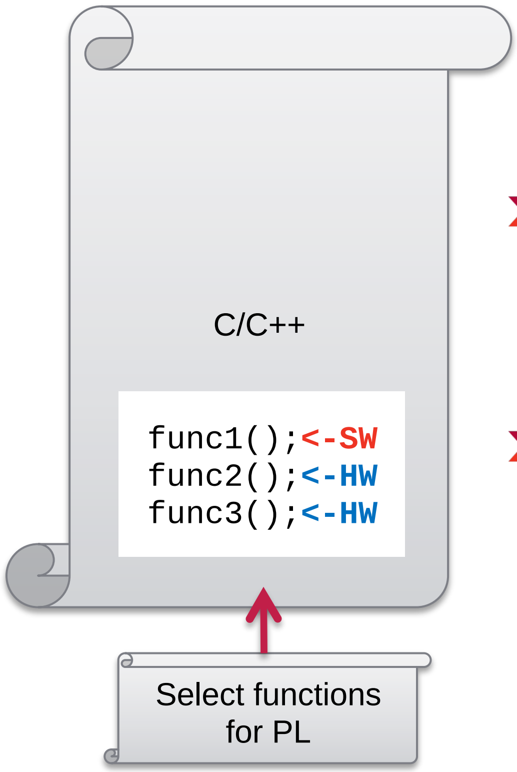
Need to modify multiple levels of design entry

After SDSoC:



- **Remove the manual design of SW drivers and HW connectivity**

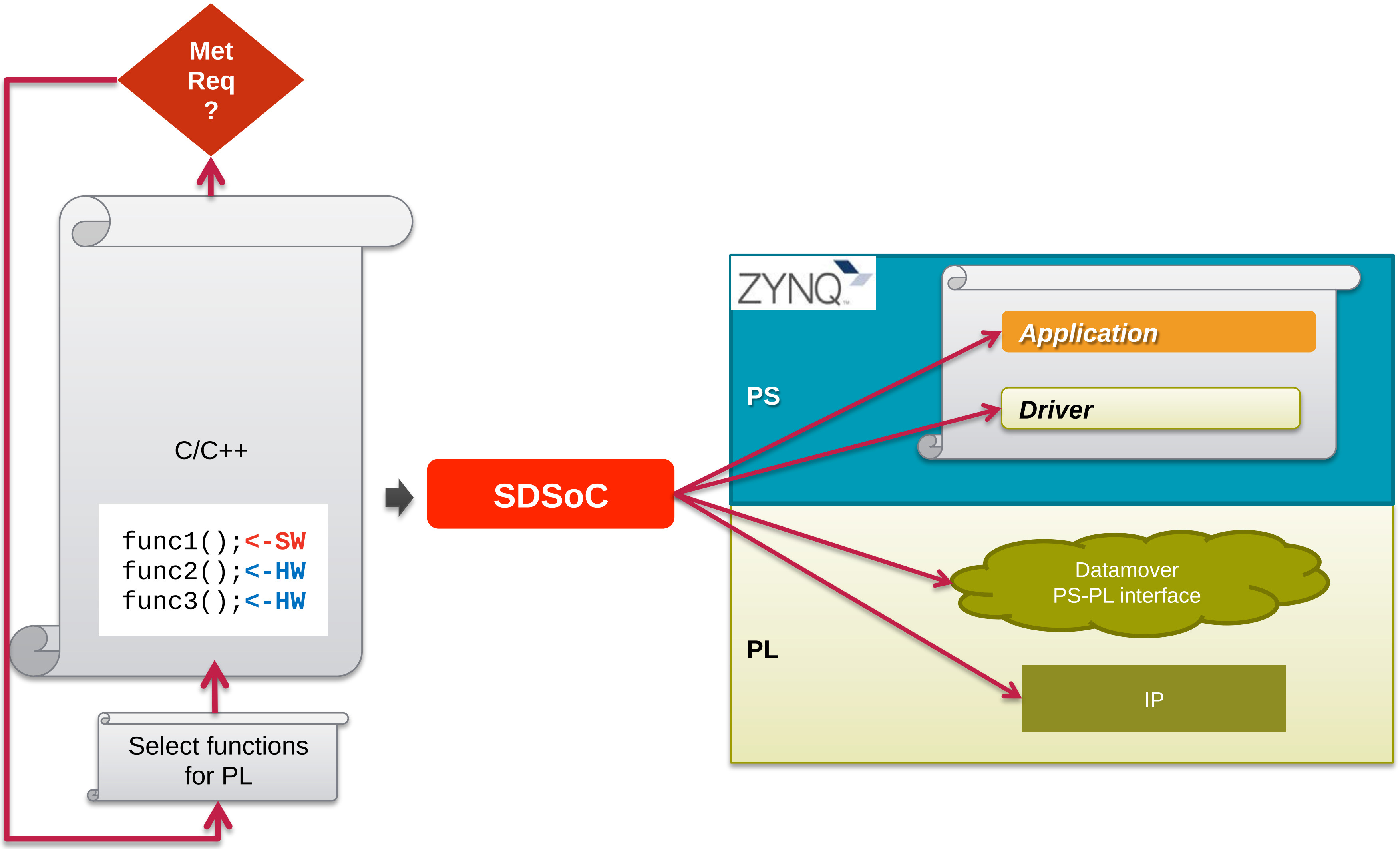
After SDSoC:



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

- **Remove the manual design of SW drivers and HW connectivity**
- **Use the C/C++ end application as the input calling the user algorithm IPs as function calls**
- **Partition set of functions to Programmable Logic by a single click**

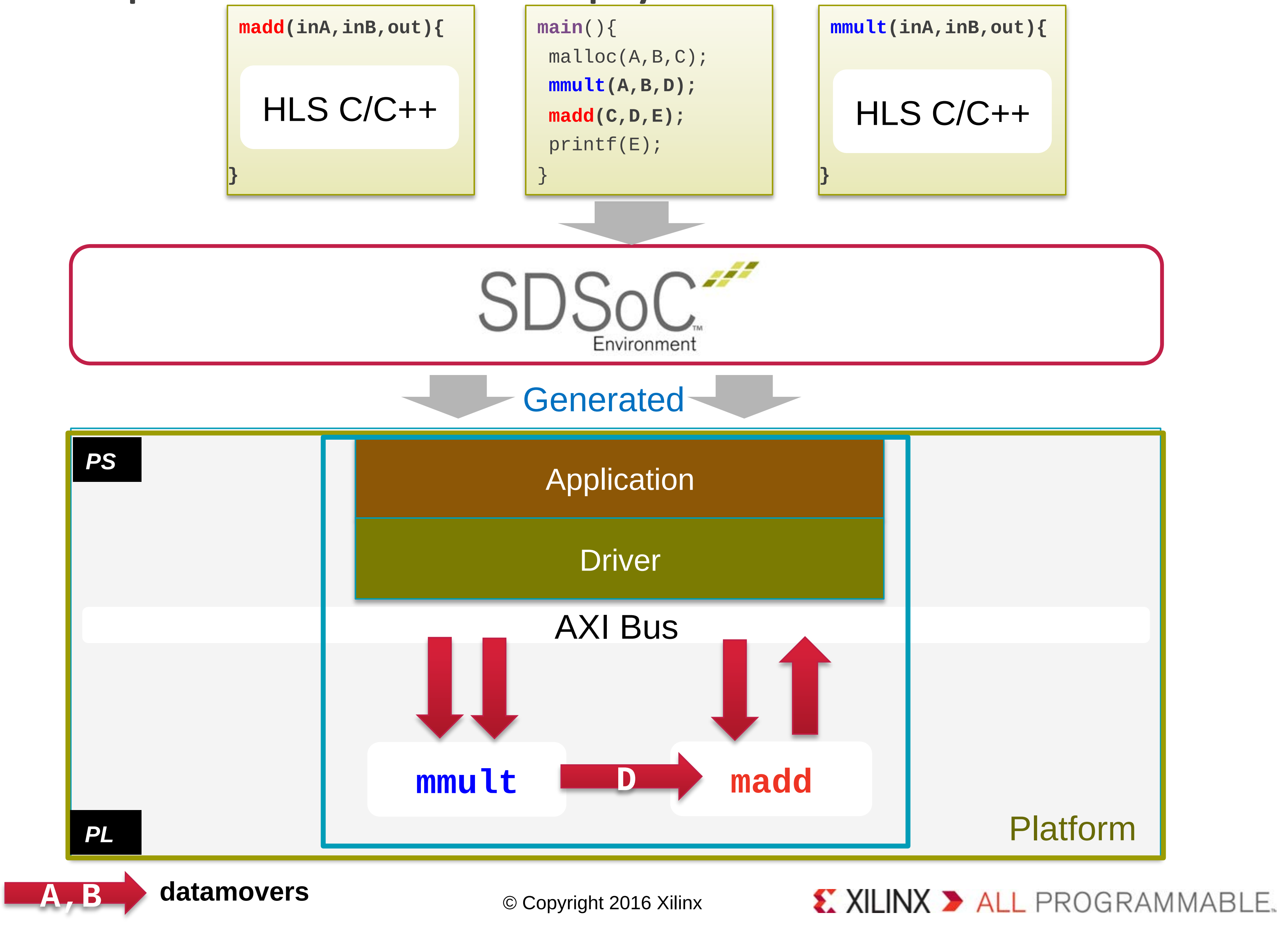
After SDSoC: Automatic System Generation



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

C/C++ to System in hours, days

Example 1: Matrix Multiply + Add



```
madd(inA,inB,out){
}
```

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

```
mmult(inA,inB,out){
}
```

XILINX ALL PROGRAMMABLE.

Example 2: 1080p60 Stereo Vision

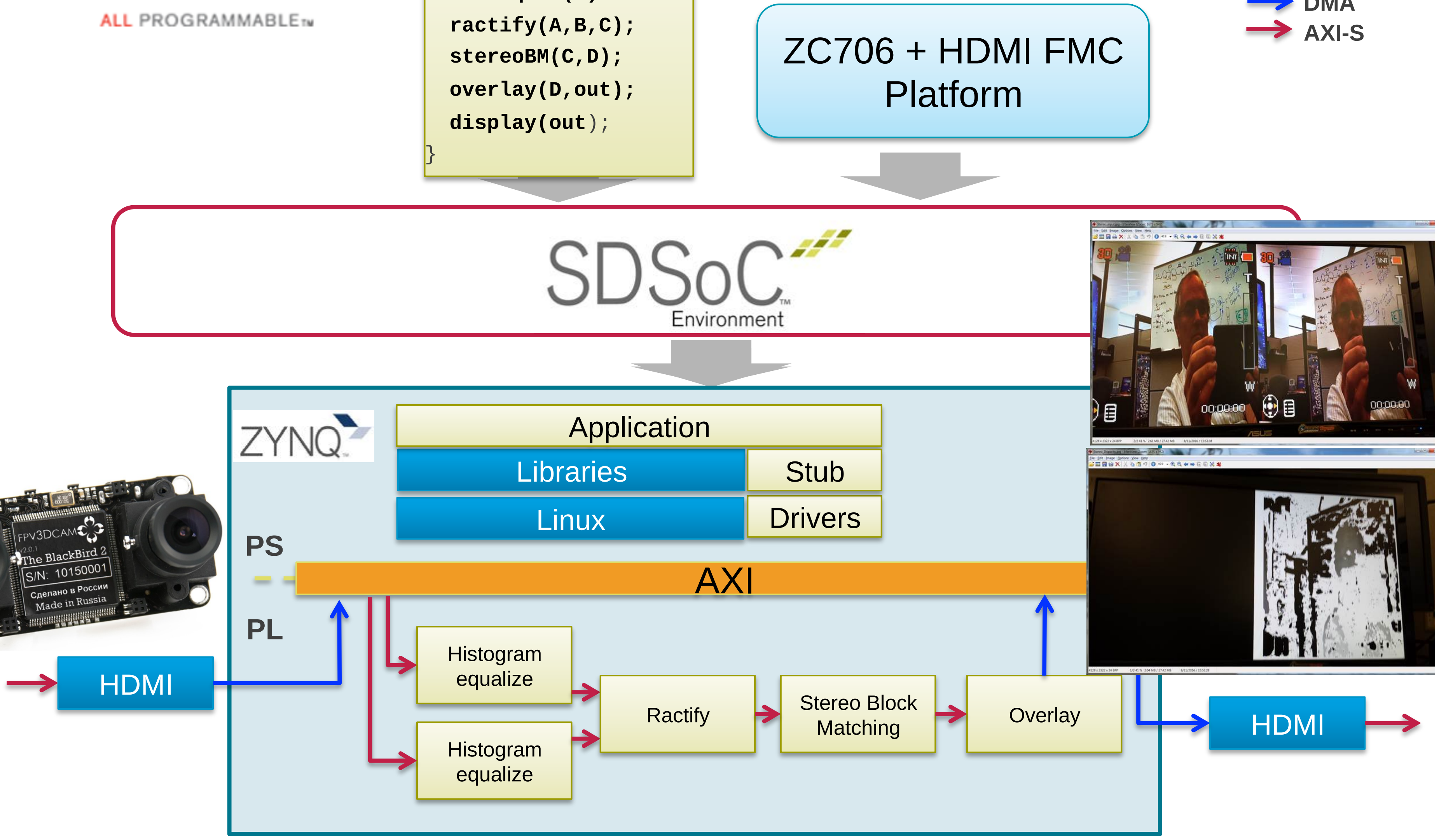
```
main(){
 histEqual(A);
 histEqual(B)
 ractify(A,B,C);
 stereoBM(C,D);
 overlay(D,out);
 display(out);
}
```

SDSoC Generated

Platform

→ DMA

→ AXI-S



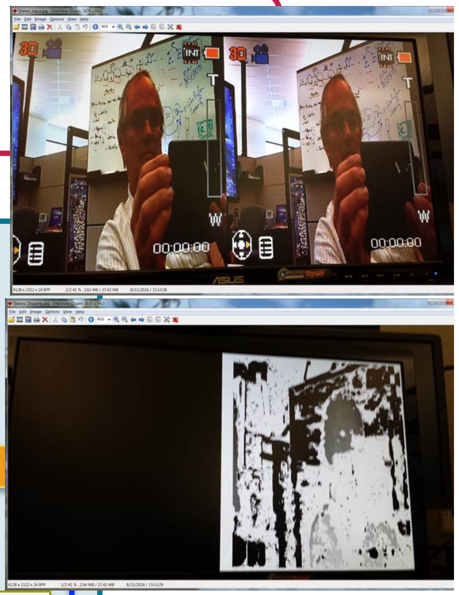


Image processing on the video I/Os via DDR3 memory

How to Call Accelerators - Programming Paradigms

- **Explicit Message Passing APIs**
  - Generic API to transfer data (send/receive, set/get)
  - Tasks written in C/C++ (SW) and/or VHDL/Verilog (HW)
  - Mental model: Threads communicating with each other

```
send_i(port1, A, …);
send_i(port2, B, …);
receive_i(port3, C, …);
…
cf_wait_all(…);
```

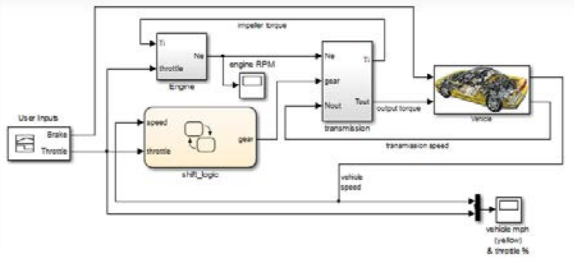
- **Function call paradigm**
  - Standard function call paradigm
    - Synchronous or asynchronous
  - Mental model: Call an accelerator that returns result

```
mm_mul(A, B, C);
// or
mm_mul_i(A, B, C, …);
…
wait(…);
```

- **Enqueue work items (OpenCL)**
  - Compile OpenCL host and kernels
  - Kernels compiled to CPU/Neon or FPGA
  - Mental model: Enqueue tasks to next available exec unit

```
*k = clCreateKernel(*prog, "mmul", &err);
err != clSetKernelArg(*k, 3, SIZE, &A);
err |= clSetKernelArg(*k, 4, SIZE, &B);
err |= clSetKernelArg(*k, 5, SIZE, &C);
err = clEnqueueNDRangeKernel(cmds, k, …);
```

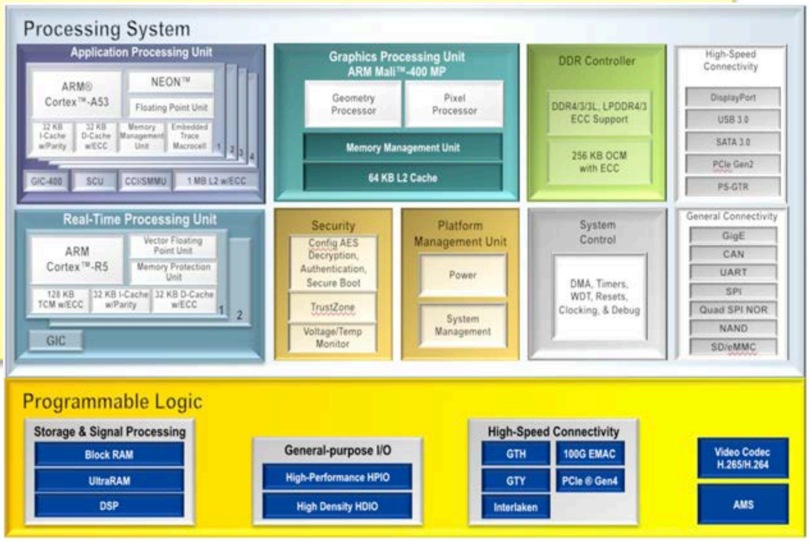
- **High level modeling**
  - MathWorks - MATLAB/Simulink
  - National Instruments – LabView



No "right" way of doing this – Depends on application

Summary

- **Heterogeneous systems are here to stay**
  - And they will be increasingly complex
- **Developing for heterogeneous systems is hard**
  - Each component might have its own language and operating environment
  - Parallel programming is hard to get right
- **New standards, tools, frameworks and APIs are here to help**
  - Hiding the complexity and unifying the environments
- **Don’t get stuck in old ways**
  - Embedded developers are conservative
  - Never a good time to try new methodologies
  - “Boiling frog” syndrome…



XILINX ALL PROGRAMMABLE.

XILINX

ALL PROGRAMMABLE™