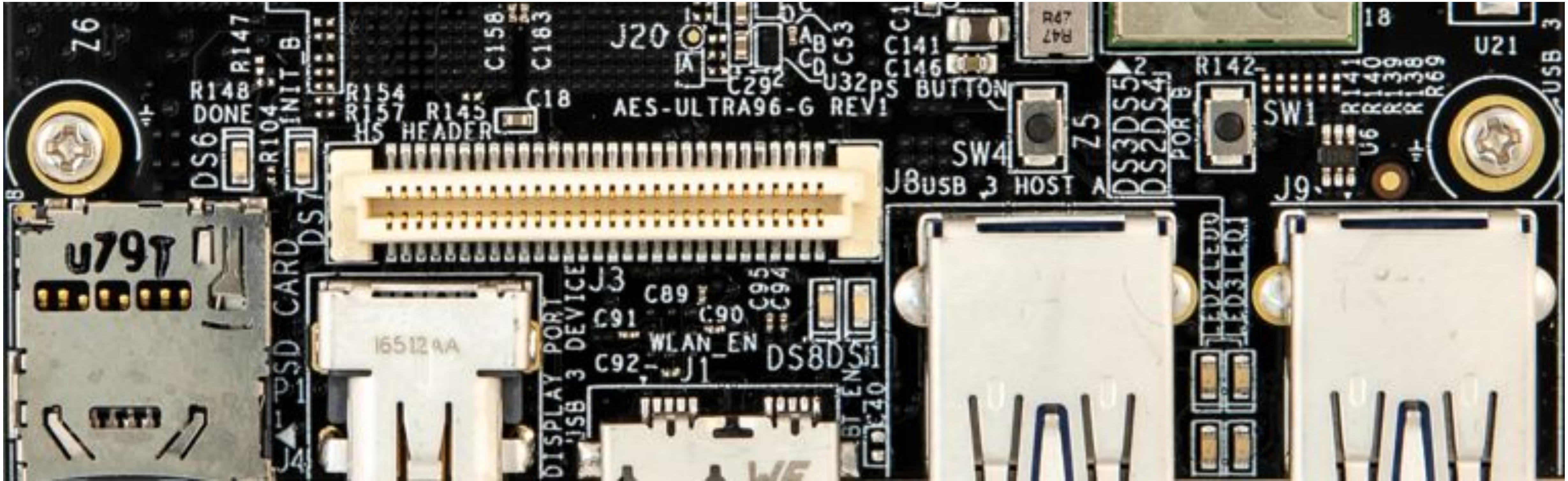
# Managing customized FPGA accelerators with SDSoC!

Linaro Conference Vancouver, CAN - 19SEP2018

AVNET®
Reach Further™



# Title: Managing customized FPGA accelerators with SDSoC!

- Abstract: Using Xilinx SDSoC and HLS as a model, this presentation, will discuss the merits and abilities of customized hardware accelerators involved with performance aspects of embedded systems. Using a designed and tested accelerator on the Avnet Ultra96, performance, power, development time metrics will be compared. Through comparing these various metrics, a new path forward for hardware accelerated software designs will be shown as a path forward for the embedded space. While this is a presentation, the author welcomes discussion!

# Agenda

- SDSoC -- What, Why, Where?
- Why is this Important to me?

--------------------------------------------------------------------------------------------

- Real Example
- Why is this faster?

--------------------------------------------------------------------------------------------

- Where do I start with SDSoC?
- What is my design flow?

--------------------------------------------------------------------------------------------

- Platforms, what are they?

- Additional Resources
- Basic HLS

# What is SDSoC?

# First Terminology

- What is HLS?
  - High Level Synthesis

- Language examples?
  - C, C++ and System C to RTL

- What is RTL?
  - Generally refers to Register Transfer Level design abstraction

- Language examples?
  - VHDL (VHSIC Hardware Description Language)
  - Verilog (IEEE 1364, is a hardware description language (HDL))

# Terminology

- Platform
  - Sandbox of resources used by SDSoC to generate solutions
  - Sometimes referred to as the “Solution Space” container
  - Defined by the target hardware

- Solution Space
  - Based on available resources
  - Same solution can be different based available resources
  - Tool will attempt to meet described performance needs

# Terminology

- PS
  - Processing System as a whole
  - Generally contains MOST hardened IP

- PL
  - Programmable Logic
  - Can be referred to as programmable fabric
  - Sometimes referred to as FPGA

- FPGA
  - Field Programmable Gate Array

# What are the Xilinx Tools?

- Vivado
  - The IDE for all Programmable Logic Design

- Supports
  - Partial Reconfiguration
  - Simulation
  - Logic Analysis
  - System Generator (for DSP)
  - HLS
  - More…

  - Details of Editions:
  - https://www.xilinx.com/products/design-tools/vivado/vivado-webpack.html

# What are the Xilinx Tools?

SDK
Software Development Kit

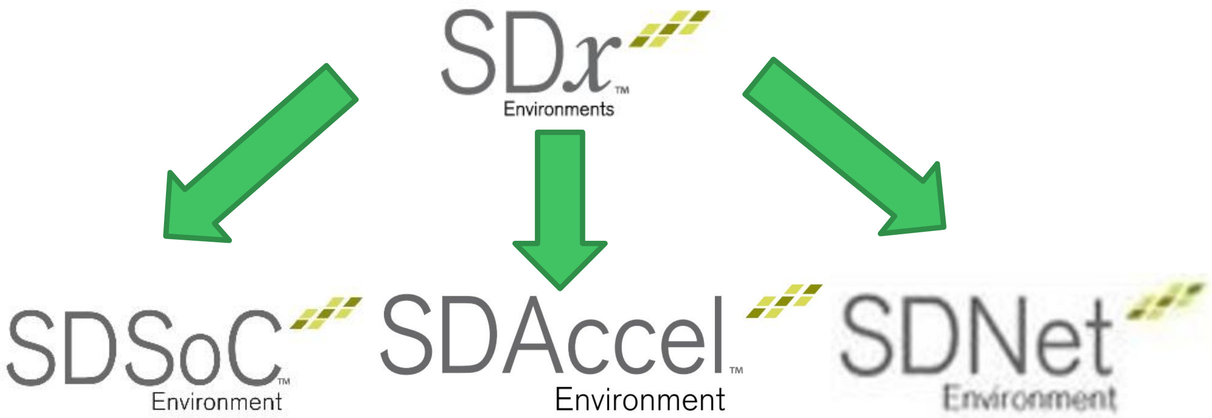
BUILT ON eclipse

- Software Development Kit
  - The base tool for all Processing System Designs

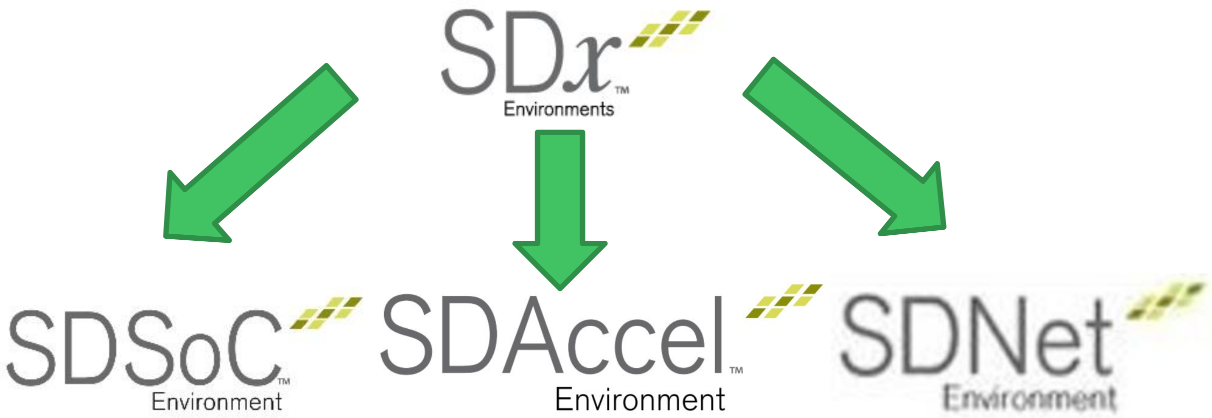
- Supports
  - Embedded Development
  - Kernel, Root File System, U-Boot
  - Xilinx provides a Yocto Build Layer
  - PetaLinux binary image creation

  - Details of Features:
  - https://www.xilinx.com/products/design-tools/software-zone/embedded-computing.html#opensource

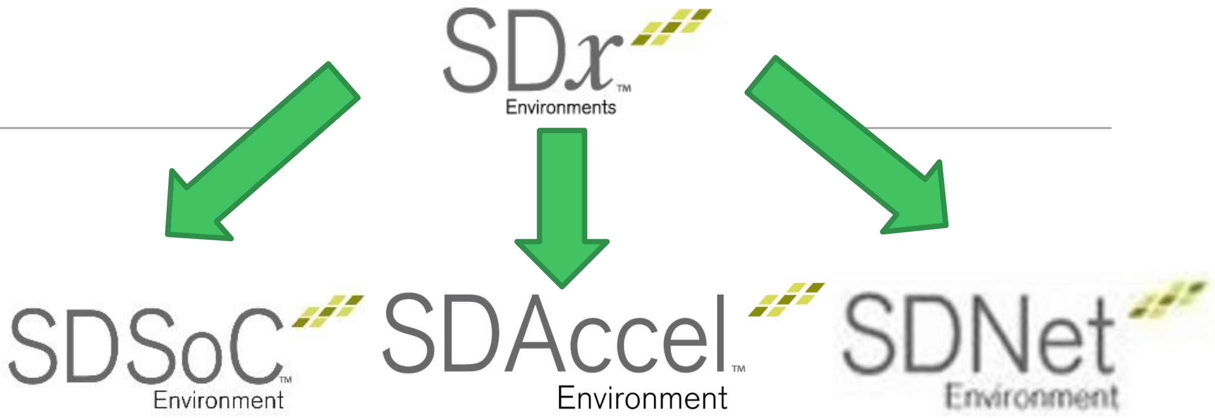
# What is SDSoC ?



# What is SDSoC ?



# What is SDSoC ?



- SDSoC is part of the SDx umbrella
- That umbrella includes:
  - SDSoC
    - Software Defined accelerators for a System on Chip
  - SDAccel
    - Software Defined Accelerators (targeting more of a PURE Programmable Logic)
  - SDNet
    - Software Defined Specification Environment for Networking

# What is SDSoC?

- SDSoC delivers
  - System level profiling
  - Automated software acceleration in programmable logic
  - Automated system connectivity generation
  - Libraries to speed programming

# What is SDSoC?

- In other words
  - It is an advanced tool suite
  - Eases the burden of managing the resources
  - Eases the burden of managing the interconnect of more complex designs
  - Deals with the complexities of tool suites for you!
  - All the while providing a better embedded design experience.

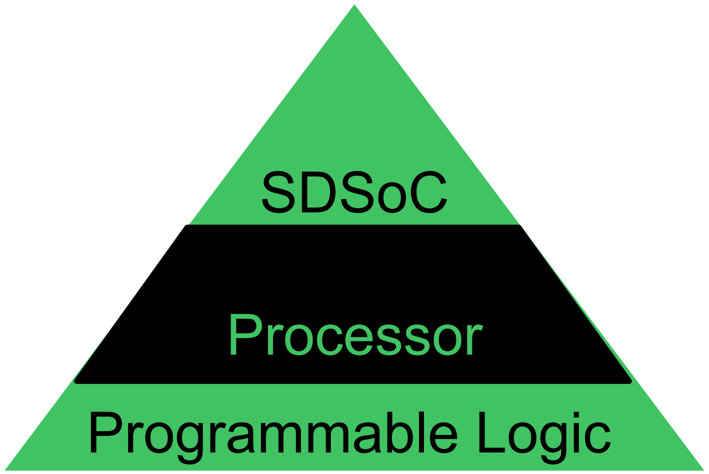
# What is SDSoC?

- So doesn’t SDSoC pick up where Vivado HLS leaves off??
  - Isn’t this just HLS?  No!
  - SDSoC is more a resource manager providing an easier path to using the suite of included tools (calling SDK, HLS, Vivado, etc.)

- What does all this mean?
  - Your job with SDSoC is to create a platform
  - You define what resources are available to the platform
  - You are to identify functions for acceleration (profiling or manually)
  - SDSoC will look at the C/C++/OpenCL and USE the available resources to most optimally hit your power/performance targets!

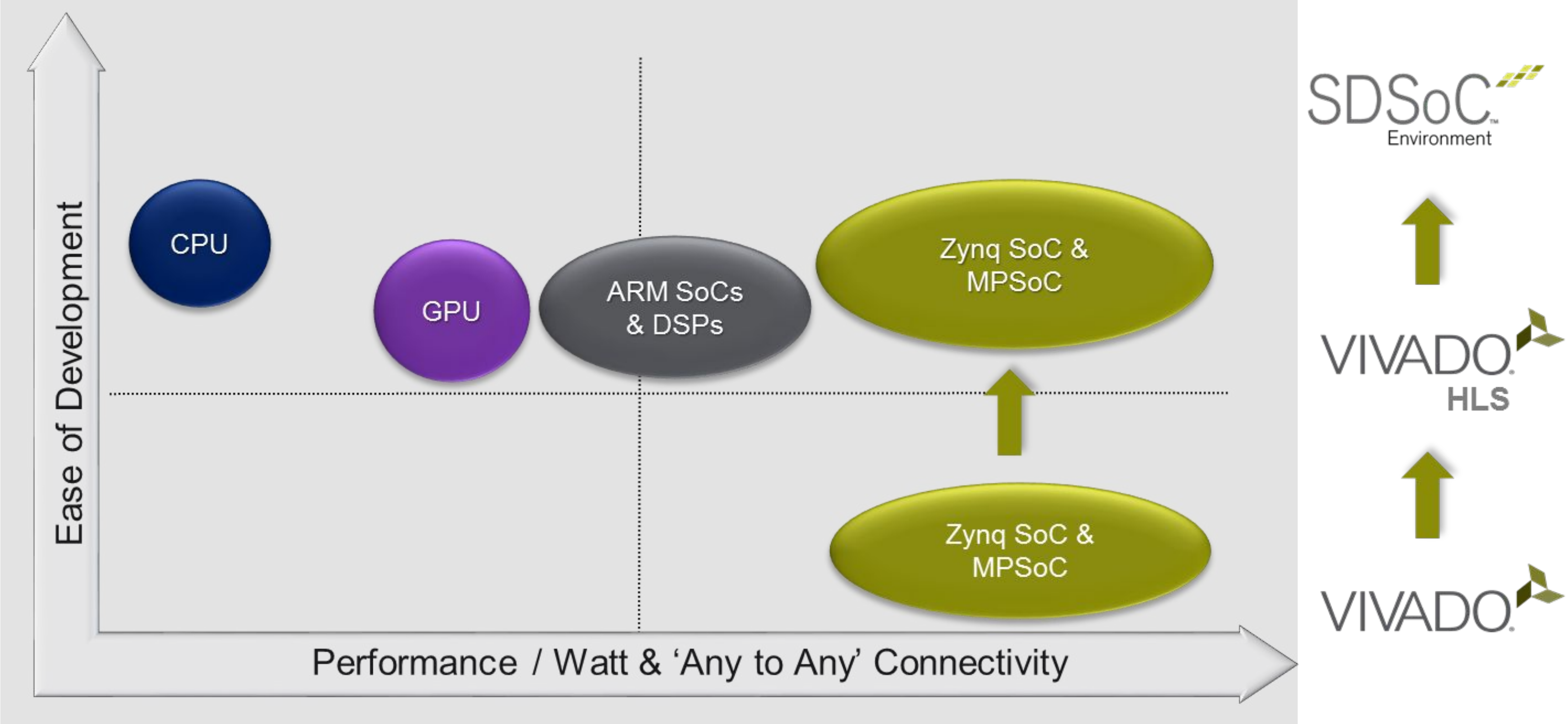
# One way to think about this…

- SDSoC works in layers
- This can be thought of as a pyramid



# Why is this important to me?

# Scaling the Productivity with Technology Advancement



Content Copyright Xilinx

# SDSoC Makes Everyone More Productive

**System Team**

- Explore HW-SW partitions and architecture rapidly in C/C++

**Hardware Team**

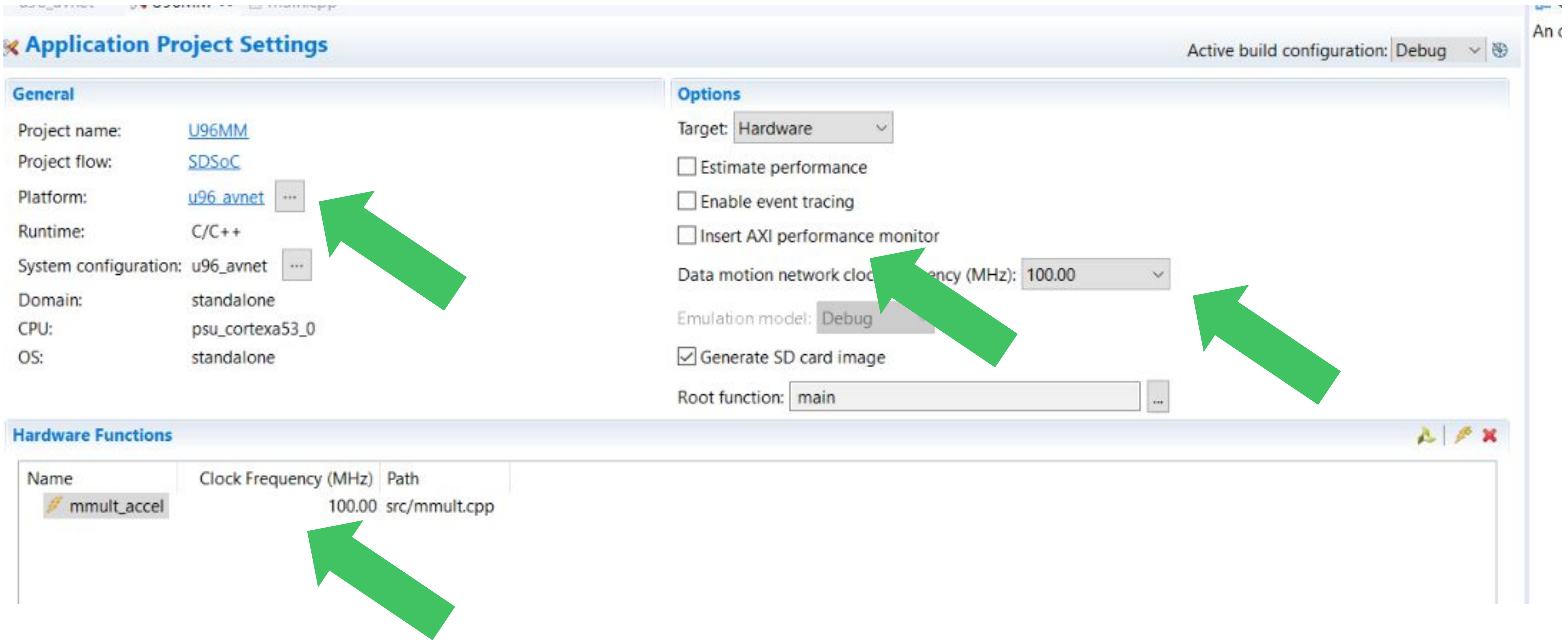
- No need to translate algorithms to HDL
- Directly optimize IP on C/C++
- Build IO system into a re-usable platform

**Software Team**

- A complete SW development environment for the full Zynq system

Content Copyright Xilinx

# SDSoC Makes Everyone More Productive



**Application Project Settings**

Active build configuration: Debug

**General**

Project name: U96MM

Project flow: SDSoC

Platform: u96_avnet ...

Runtime: C/C++

System configuration: u96_avnet ...

Domain: standalone

CPU: psu_cortexa53_0

OS: standalone

**Options**

Target: Hardware

☐ Estimate performance

☐ Enable event tracing

☐ Insert AXI performance monitor

Data motion network clock frequency (MHz): 100.00

Emulation model: Debug

☑ Generate SD card image

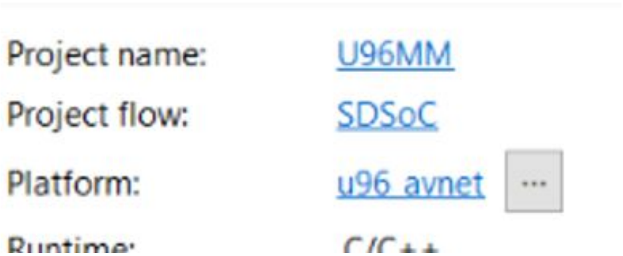
Root function: main ...

**Hardware Functions**

| Name | Clock Frequency (MHz) | Path |
|---|---|---|
| mmult_accel | 100.00 | src/mmult.cpp |

Content Copyright Xilinx

# SDSoC Makes Everyone More Productive



Project name: U96MM

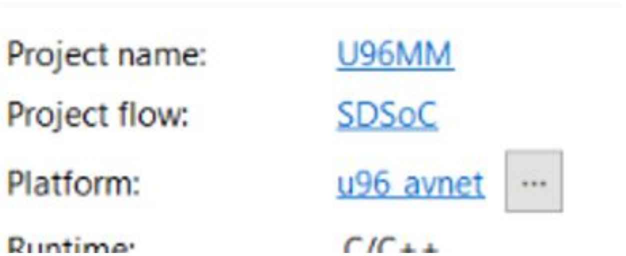
Project flow: SDSoC

Platform: u96_avnet

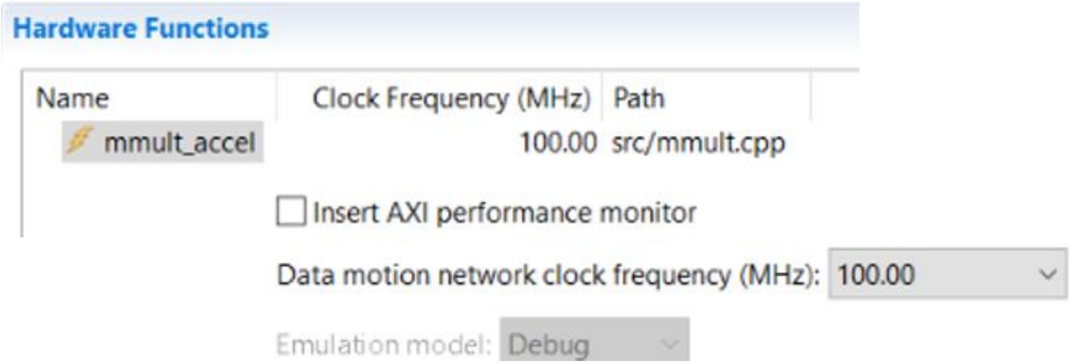
Runtime: C/C++

- By working at a system level in the solution space
  - Need only create the solution once
  - You can retarget to a new platform with different resources with a simple retarget

# SDSoC Makes Everyone More Productive



| | |
|---|---|
| Project name: | U96MM |
| Project flow: | SDSoC |
| Platform: | u96_avnet |
| Runtime: | C/C++ |



Hardware Functions

| Name | Clock Frequency (MHz) | Path |
|---|---|---|
| mmult_accel | 100.00 | src/mmult.cpp |

☐ Insert AXI performance monitor

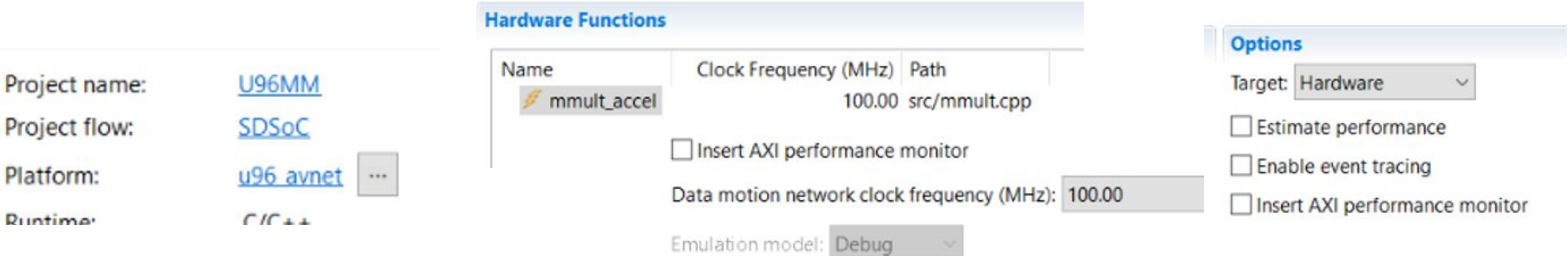
Data motion network clock frequency (MHz): 100.00

Emulation model: Debug

- By working at a system level in the solution space
  - Need only create the solution once
  - You can retarget to a new platform with different resources with a simple retarget
  - You can also quickly and easily change performance behaviors through a few selections

Content Copyright Xilinx

# SDSoC Makes Everyone More Productive



Project name: U96MM

Project flow: SDSoC

Platform: u96_avnet ...

Runtime: C/C++

**Hardware Functions**

| Name | Clock Frequency (MHz) | Path |
|---|---|---|
| mmult_accel | 100.00 | src/mmult.cpp |

□ Insert AXI performance monitor

Data motion network clock frequency (MHz): 100.00

Emulation model: Debug

**Options**

Target: Hardware

□ Estimate performance

□ Enable event tracing

□ Insert AXI performance monitor

- By working at a system level in the solution space
  - Need only create the solution once
  - You can retarget to a new platform with different resources with a simple retarget
  - You can also quickly and easily change performance behaviors through a few selections
  - Lastly, there is a Performance Estimation tool which will significantly help you target WHICH algorithms should be targeted to hardware acceleration

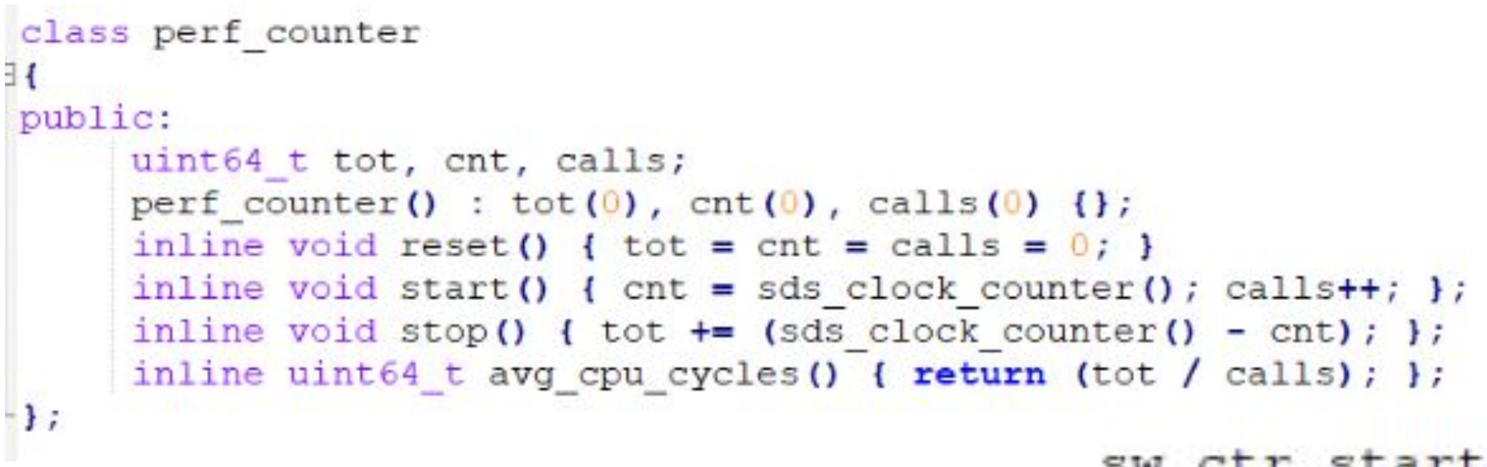
# …Why is this important to me?

- You do not have to be a FPGA expert!
  - SDSoC is another step closer to more easily empowering Engineers who have little to no FPGA/HDL experience

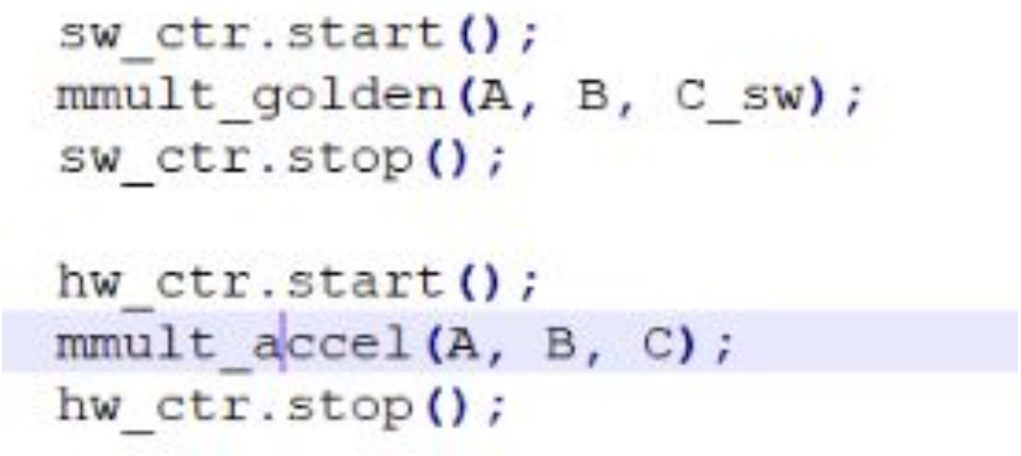
- Problems are more complex
  - SDSoC allows a designer to take a system’s approach
  - SDSoC allows you to import your existing Zynq design and start developing new applications in C/C++/OpenCL
  - Designers can concentrate more effort on the solution
    - Instead of determining gate level logic

# Real Example

# How about a REAL example?

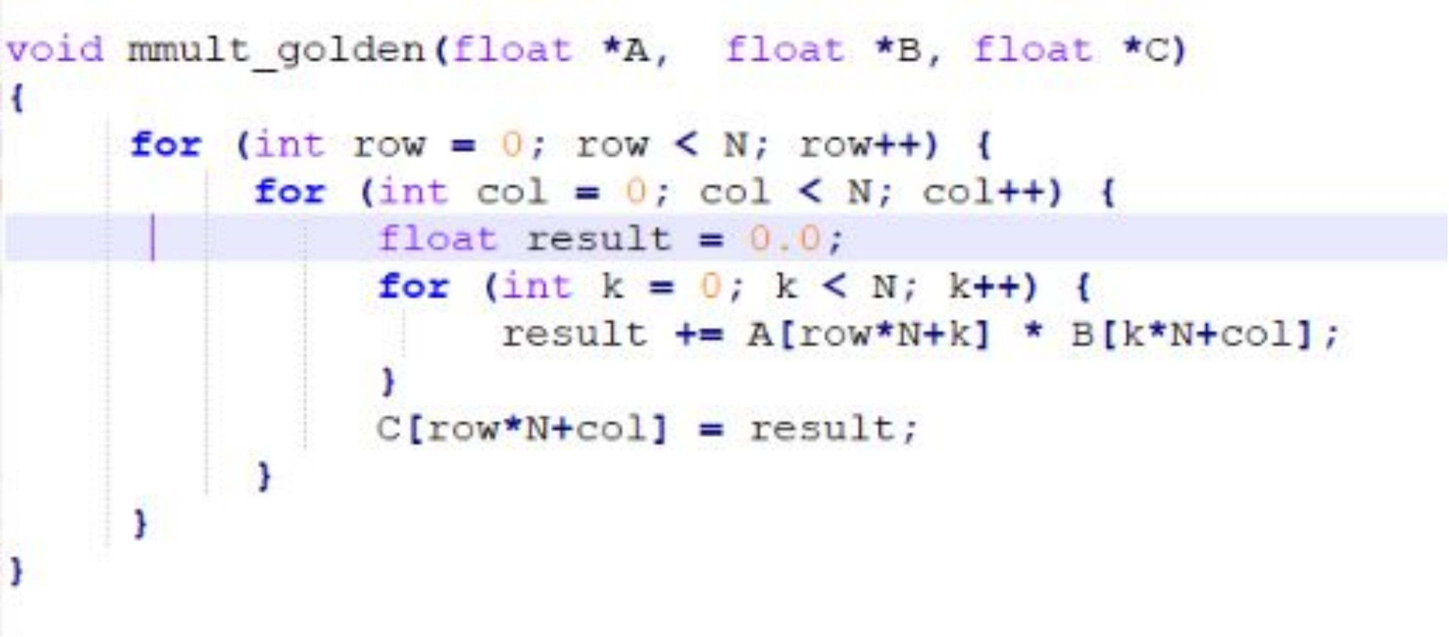


```
class perf_counter
{
public:
        uint64_t tot, cnt, calls;
        perf_counter() : tot(0), cnt(0), calls(0) {};
        inline void reset() { tot = cnt = calls = 0; }
        inline void start() { cnt = sds_clock_counter(); calls++; };
        inline void stop() { tot += (sds_clock_counter() - cnt); };
        inline uint64_t avg_cpu_cycles() { return (tot / calls); };
};
```



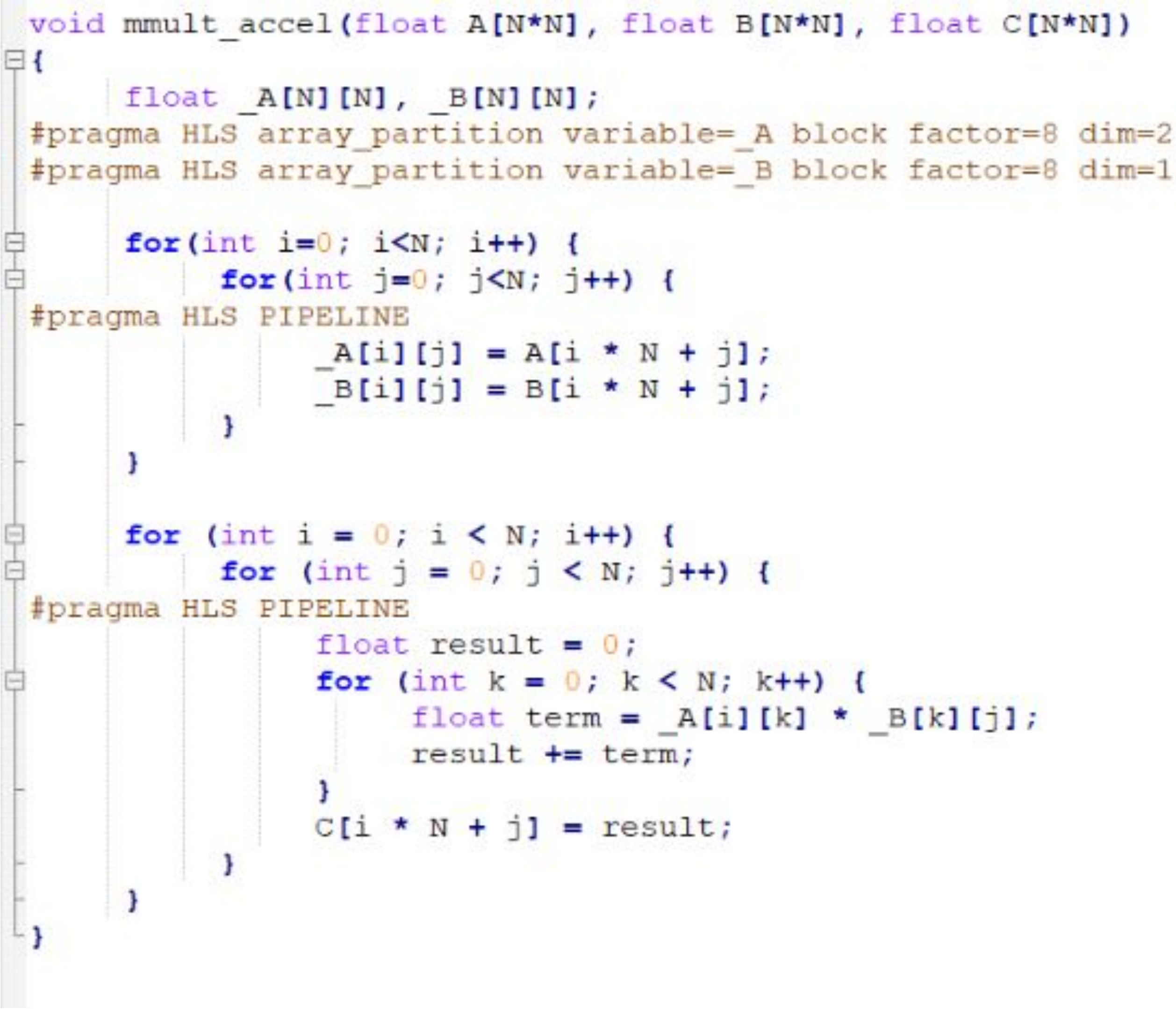
```
sw_ctr.start();
mmult_golden(A, B, C_sw);
sw_ctr.stop();

hw_ctr.start();
mmult_accel(A, B, C);
hw_ctr.stop();
```

# How about a REAL example?



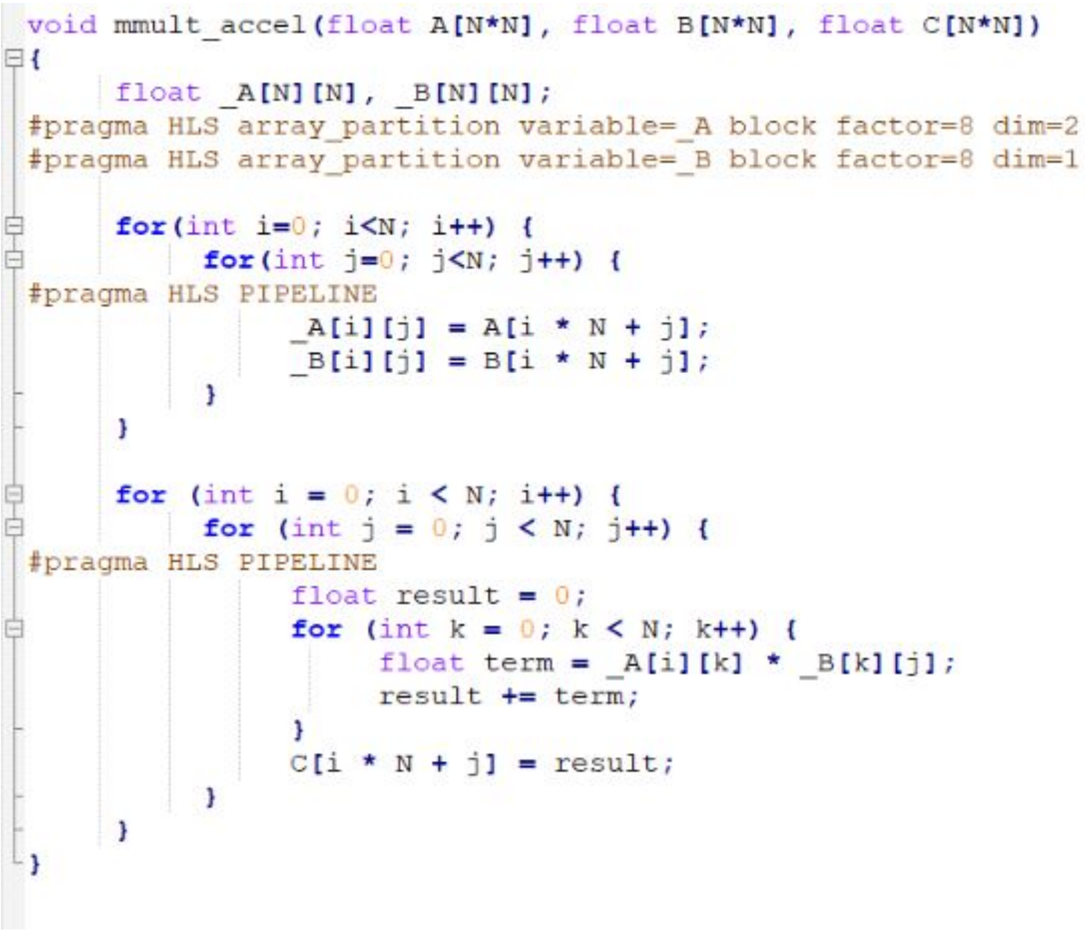
```c
void mmult_golden(float *A,  float *B, float *C)
{
     for (int row = 0; row < N; row++) {
          for (int col = 0; col < N; col++) {
               float result = 0.0;
               for (int k = 0; k < N; k++) {
                    result += A[row*N+k] * B[k*N+col];
               }
               C[row*N+col] = result;
          }
     }
}
```

# How about a REAL example?

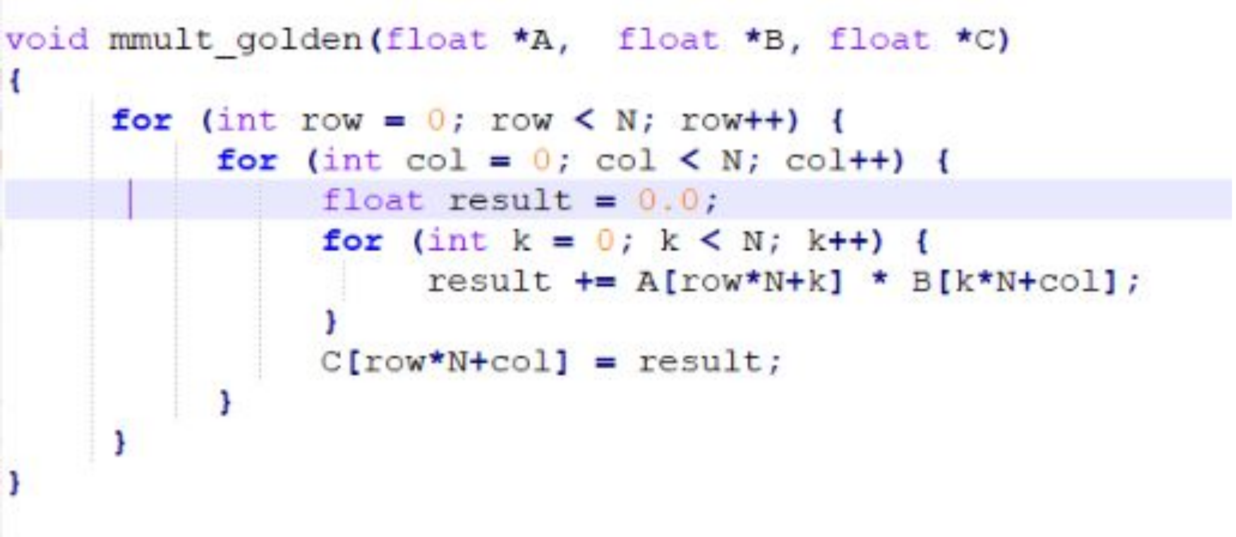


```
void mmult_accel(float A[N*N], float B[N*N], float C[N*N])
{
    float _A[N][N], _B[N][N];
#pragma HLS array_partition variable=_A block factor=8 dim=2
#pragma HLS array_partition variable=_B block factor=8 dim=1

    for(int i=0; i<N; i++) {
        for(int j=0; j<N; j++) {
#pragma HLS PIPELINE
            _A[i][j] = A[i * N + j];
            _B[i][j] = B[i * N + j];
        }
    }

    for (int i = 0; i < N; i++) {
        for (int j = 0; j < N; j++) {
#pragma HLS PIPELINE
            float result = 0;
            for (int k = 0; k < N; k++) {
                float term = _A[i][k] * _B[k][j];
                result += term;
            }
            C[i * N + j] = result;
        }
    }
}
```

# How about a REAL example?



```
void mmult_accel(float A[N*N], float B[N*N], float C[N*N])
{
    float _A[N][N], _B[N][N];
#pragma HLS array_partition variable=_A block factor=8 dim=2
#pragma HLS array_partition variable=_B block factor=8 dim=1

    for(int i=0; i<N; i++) {
        for(int j=0; j<N; j++) {
#pragma HLS PIPELINE
            _A[i][j] = A[i * N + j];
            _B[i][j] = B[i * N + j];
        }
    }

    for (int i = 0; i < N; i++) {
        for (int j = 0; j < N; j++) {
#pragma HLS PIPELINE
            float result = 0;
            for (int k = 0; k < N; k++) {
                float term = _A[i][k] * _B[k][j];
                result += term;
            }
            C[i * N + j] = result;
        }
    }
}
```



```
void mmult_golden(float *A,  float *B, float *C)
{
    for (int row = 0; row < N; row++) {
        for (int col = 0; col < N; col++) {
            float result = 0.0;
            for (int k = 0; k < N; k++) {
                result += A[row*N+k] * B[k*N+col];
            }
            C[row*N+col] = result;
        }
    }
}
```

# How about some REAL numbers?

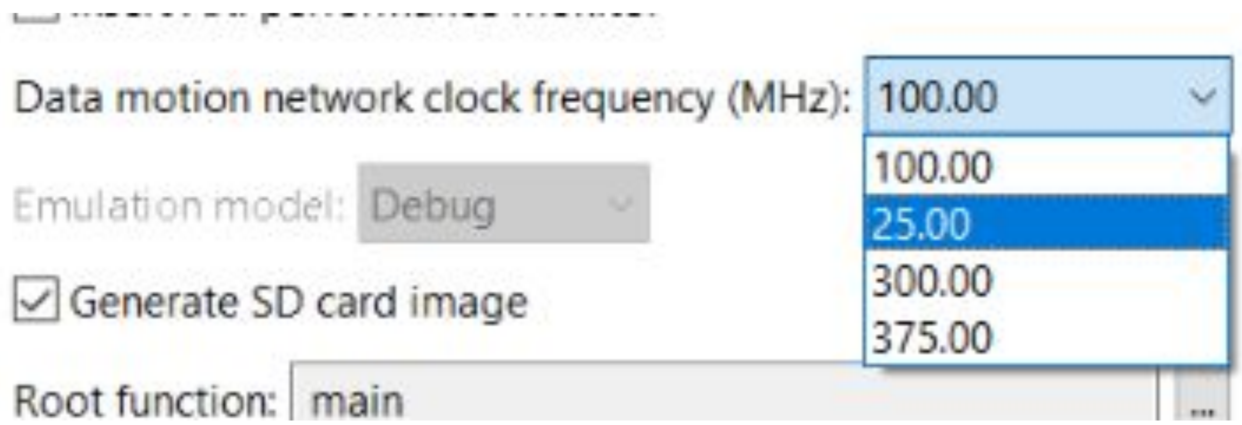
- When working in the Processor, typically we use FLOPS.
  - The A-53 processors in this design are configured for **1200MHz**

- When working in the Programmable Logic, typically we use CLOCKS.
  - The Programmable Logic is running at **100MHz, 300MHz, 375MHz**

Why the difference?
PS has many pipelines, caches, word execution times, etc., which can confuse the number of clock cycles
PL runs on an exact cadence, all pipelines, etc. can be exactly modeled as YOU CREATED THEM!

# How about some REAL numbers?

- PC: I7- 4 Core 3.17GHz; 48GB Ram; SSD
- Platform Build Times?
  - Vivado 11:30
  - DSA Creation 0:10
  - SDx 4:00
- Solution Build Times?
  - Program Setup0:33
  - For 100MHz PL 35:34 (1st Run)
  - For 100MHz PL 24:34 (2nd Run)
  - For 300MHz PL 35:54
  - For 375MHz PL 42:22



Data motion network clock frequency (MHz): 100.00

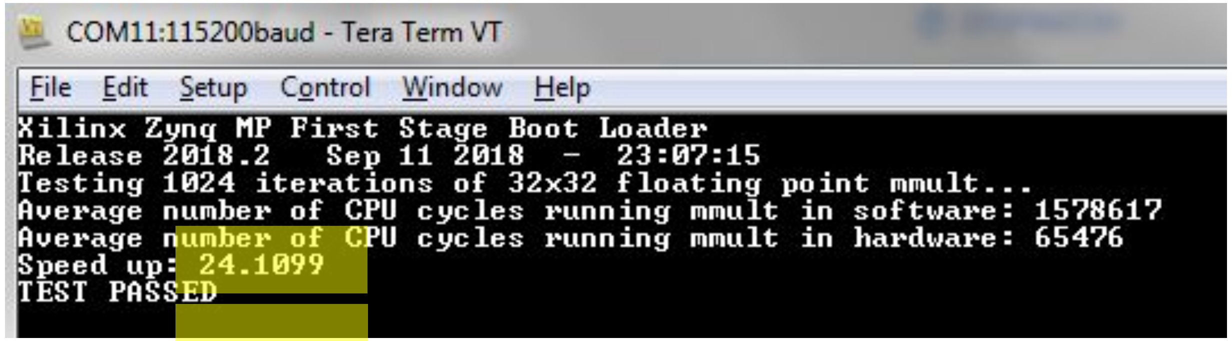
100.00
25.00
300.00
375.00

Emulation model: Debug

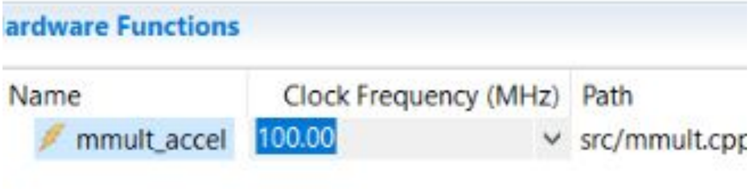
☑ Generate SD card image

Root function: main

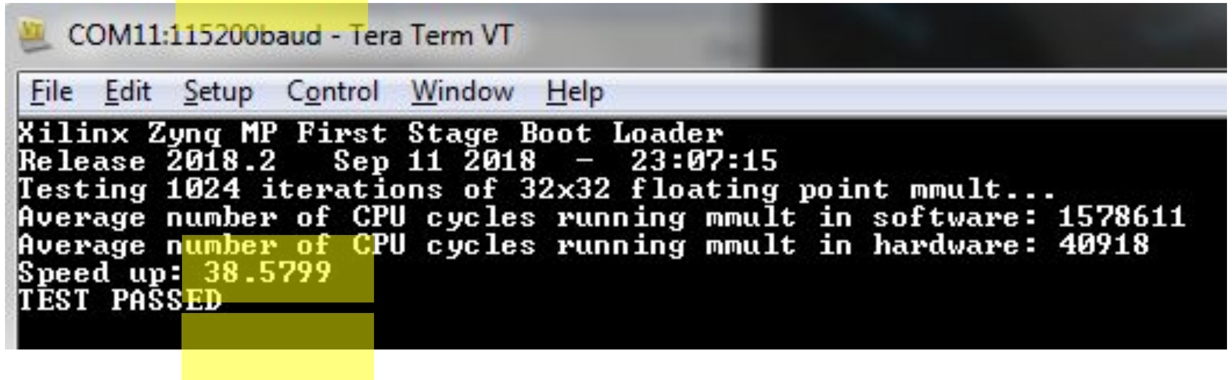
# How about some REAL numbers?


COM11:115200baud - Tera Term VT
File Edit Setup Control Window Help

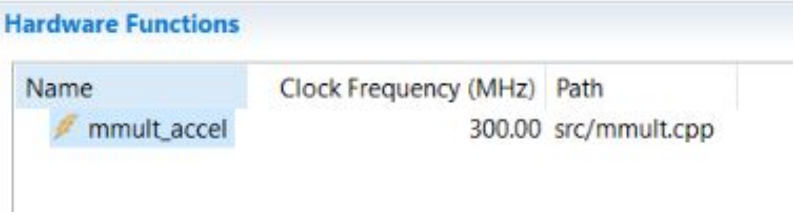
```
Xilinx Zynq MP First Stage Boot Loader
Release 2018.2   Sep 11 2018  -  23:07:15
Testing 1024 iterations of 32x32 floating point mmult...
Average number of CPU cycles running mmult in software: 1578617
Average number of CPU cycles running mmult in hardware: 65476
Speed up: 24.1099
TEST PASSED
```


ardware Functions

| Name | Clock Frequency (MHz) | Path |
|---|---|---|
| mmult_accel | 100.00 | src/mmult.cpp |

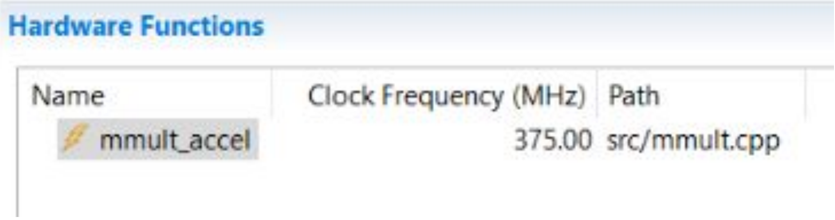

COM11:115200baud - Tera Term VT
File Edit Setup Control Window Help

```
Xilinx Zynq MP First Stage Boot Loader
Release 2018.2   Sep 11 2018  -  23:07:15
Testing 1024 iterations of 32x32 floating point mmult...
Average number of CPU cycles running mmult in software: 1578611
Average number of CPU cycles running mmult in hardware: 40918
Speed up: 38.5799
TEST PASSED
```


Hardware Functions

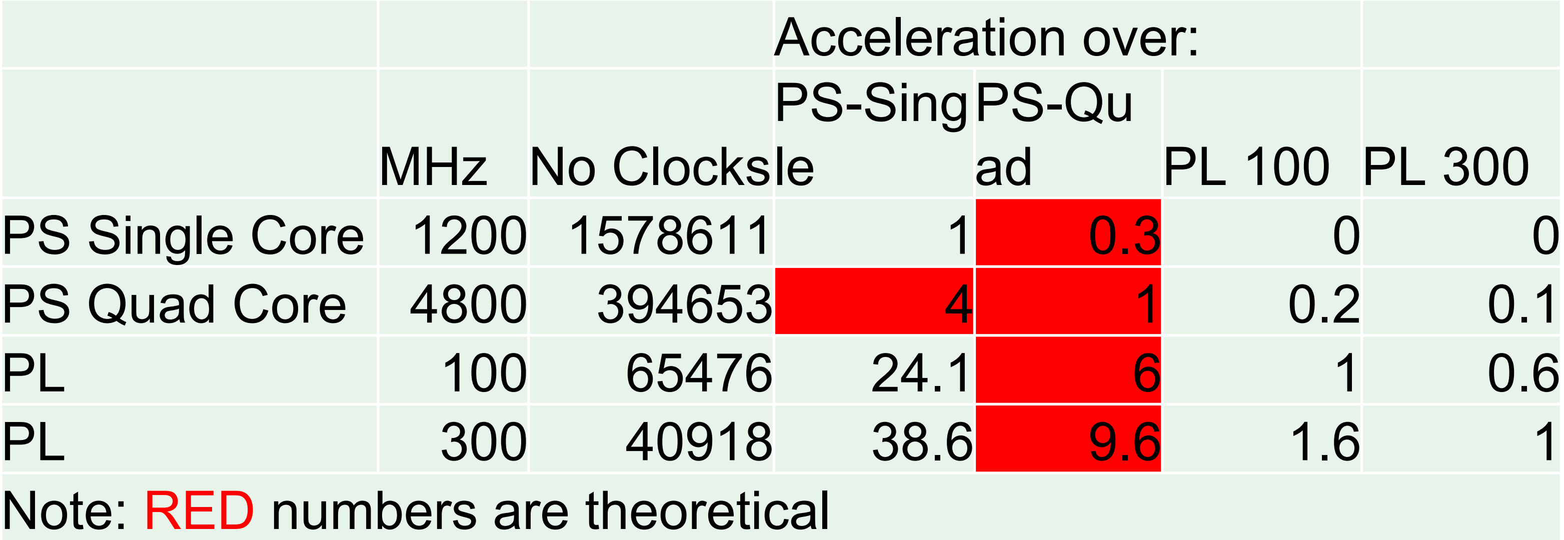
| Name | Clock Frequency (MHz) | Path |
|---|---|---|
| mmult_accel | 300.00 | src/mmult.cpp |

Did not meet timing


Hardware Functions

| Name | Clock Frequency (MHz) | Path |
|---|---|---|
| mmult_accel | 375.00 | src/mmult.cpp |

# Data – in detail



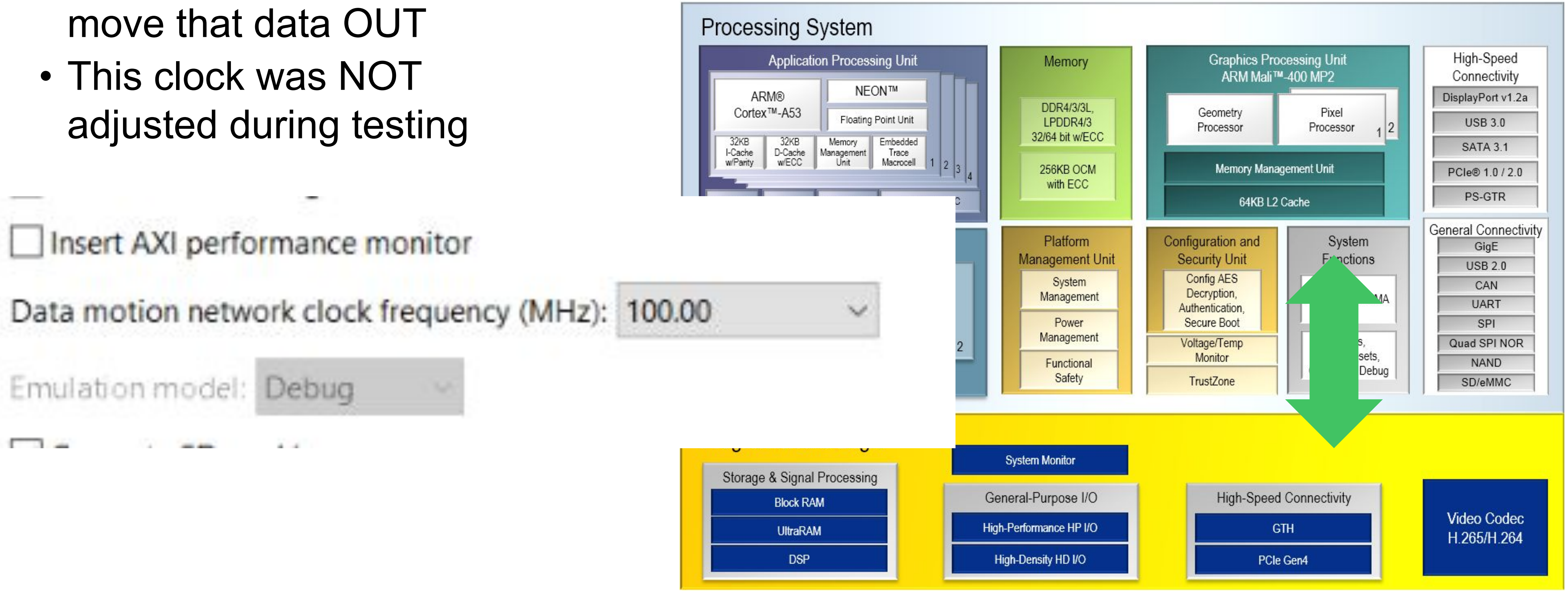
| | | | Acceleration over: | | | |
|---|---|---|---|---|---|---|
| | MHz | No Clocks | PS-Single | PS-Quad | PL 100 | PL 300 |
| PS Single Core | 1200 | 1578611 | 1 | 0.3 | 0 | 0 |
| PS Quad Core | 4800 | 394653 | 4 | 1 | 0.2 | 0.1 |
| PL | 100 | 65476 | 24.1 | 6 | 1 | 0.6 |
| PL | 300 | 40918 | 38.6 | 9.6 | 1.6 | 1 |

Note: RED numbers are theoretical

- To perform 1024 cycles of a 32x32 matrix multiply
  - The PS takes 1,578,611 clocks @ 1200MHz
  - The PL takes 65,476 PS clocks @ 100MHz
  - The PL takes 40,918 PS clocks @ 300MHz
  - 24x increase with PL @ 100MHz
  - ~39x increase with PL @ 300MHz
  - 1.6x increase with PL 100→300MHz

# Why is it not Linear?

- If we process 3x faster, would we not see 3x improvement in our calculation?
- Remember that we have to move the data INTO the programmable logic as well as move that data OUT
- This clock was NOT adjusted during testing



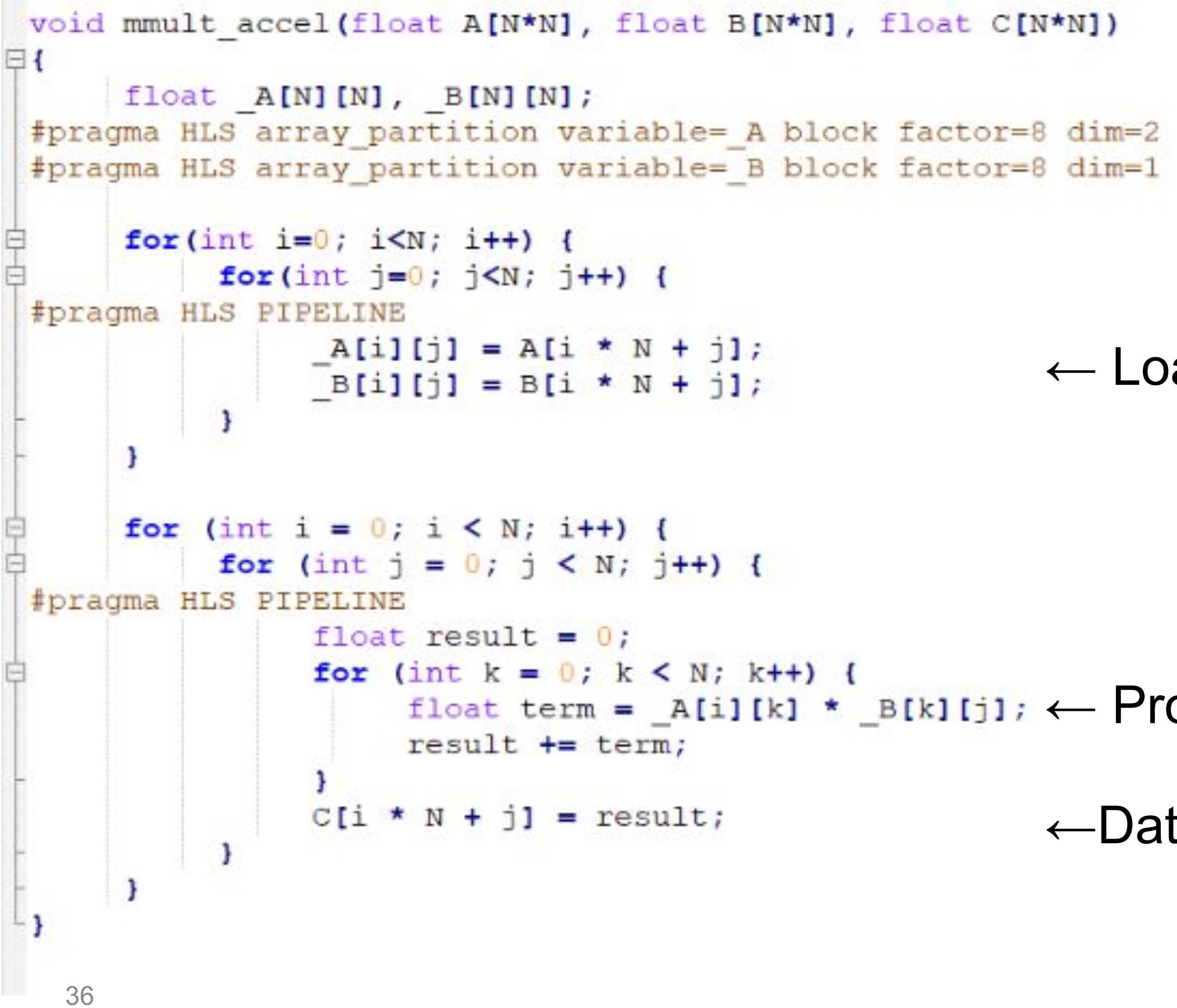
☐ Insert AXI performance monitor

Data motion network clock frequency (MHz): 100.00

Emulation model: Debug

# Why is this faster?

# Parallel Execution



```
void mmult_accel(float A[N*N], float B[N*N], float C[N*N])
{
    float _A[N][N], _B[N][N];
#pragma HLS array_partition variable=_A block factor=8 dim=2
#pragma HLS array_partition variable=_B block factor=8 dim=1

    for(int i=0; i<N; i++) {
        for(int j=0; j<N; j++) {
#pragma HLS PIPELINE
            _A[i][j] = A[i * N + j];
            _B[i][j] = B[i * N + j];
        }
    }

    for (int i = 0; i < N; i++) {
        for (int j = 0; j < N; j++) {
#pragma HLS PIPELINE
            float result = 0;
            for (int k = 0; k < N; k++) {
                float term = _A[i][k] * _B[k][j];
                result += term;
            }
            C[i * N + j] = result;
        }
    }
}
```

← Loading Arrays into Programmable Logic

← Programmable Logic Executes in parallel

←Datamover used to place into DDR4

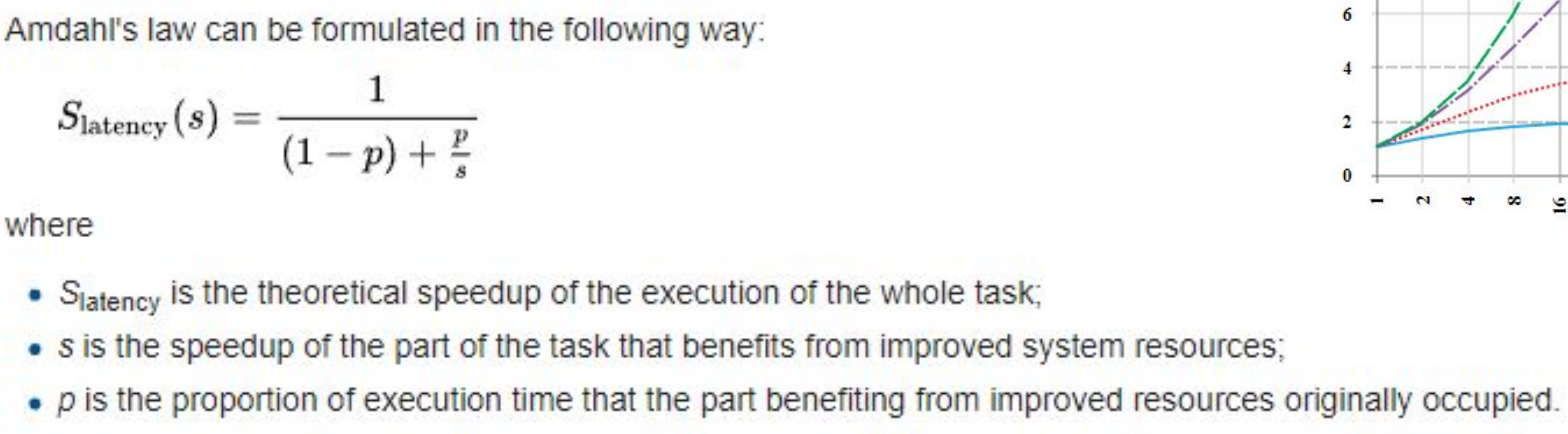
# Parallel Execution – Limitations?

- By breaking up algorithms into smaller pieces
  - How parallel does your algorithm support?

- Depending on physical limitations
  - power density of processing unit
  - transistor count
  - frequency limitations (just like our 375MHz case)

# Amdhal's Law

- My take
  - Due to losses in efficiency of parallelism as well as limitations on the slowest part, an ever increasing number of processing units is needed to maintain your speedup

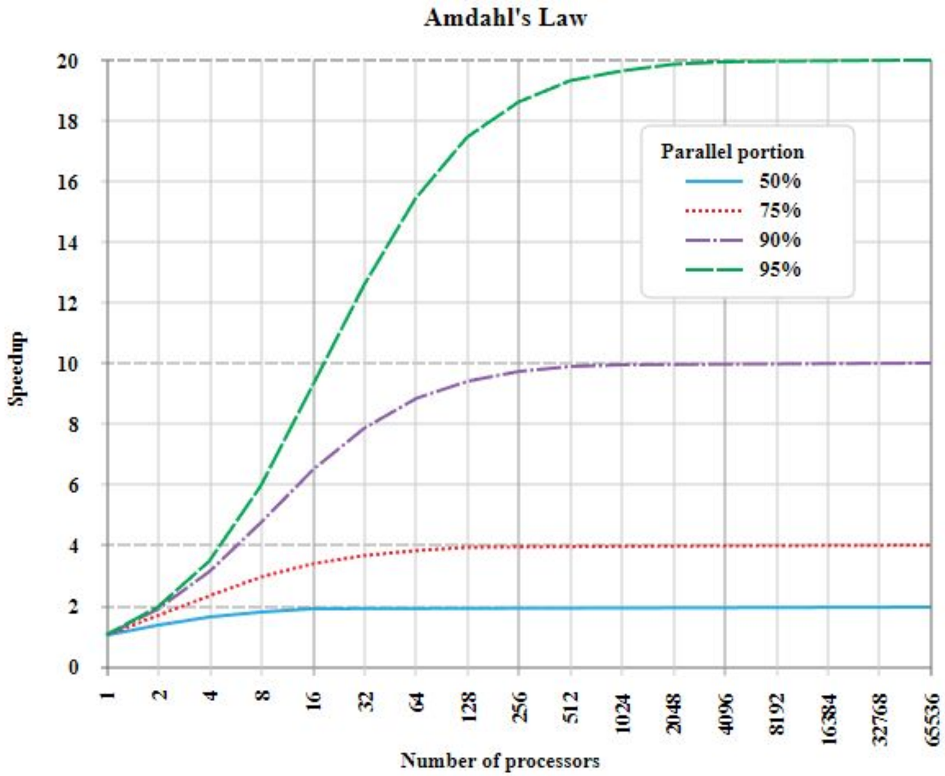


Amdahl's law can be formulated in the following way:

$$S_{\text{latency}}(s) = \frac{1}{(1-p) + \frac{p}{s}}$$

where

- $S_{\text{latency}}$ is the theoretical speedup of the execution of the whole task;
- $s$ is the speedup of the part of the task that benefits from improved system resources;
- $p$ is the proportion of execution time that the part benefiting from improved resources originally occupied.



https://en.wikipedia.org/wiki/Amdahl%27s_law

# Parallel Execution – Programmable Fabric Shines!

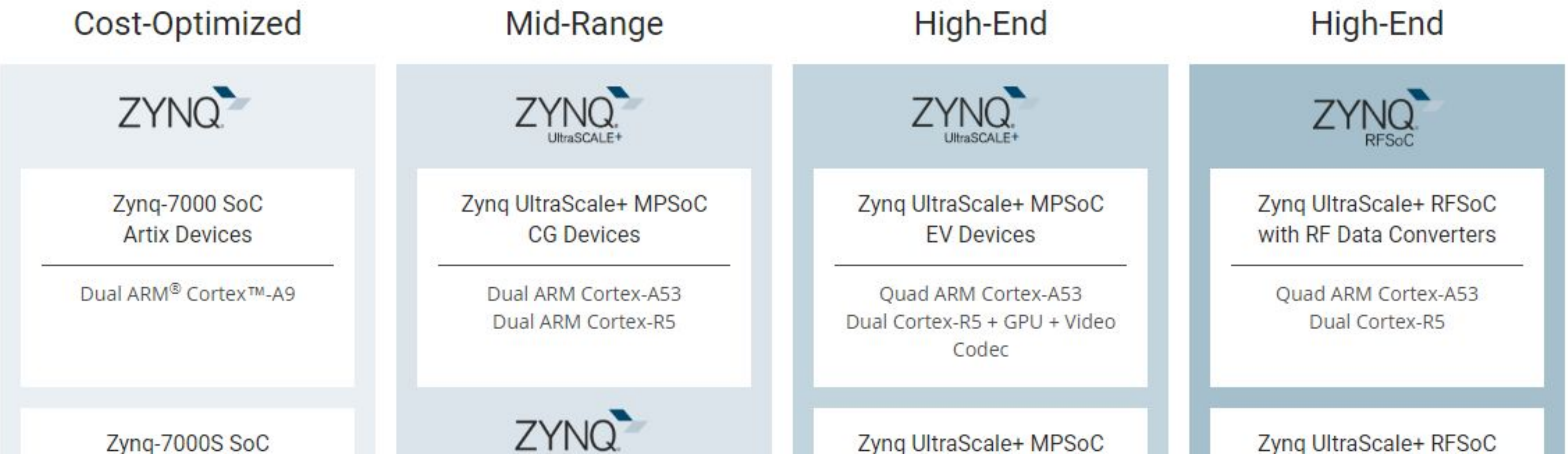
- In the case of Programmable Logic
  - We break the problem up into the smallest pieces
  - Create a customized, dedicated accelerator
  - Providing resources specifically to handle the problem

- In the Matrix Multiply
  - Ideally we would have 1 multiplier for EACH multiplication that is needed as well as an adder for EACH resultant, providing a 2 to 3 PL clock result (~20-30 PS clocks)

- In our SDSoC solution
  - The tool evaluated the resources we had at hand and provided enough resources to get us down to ~40 – 64 PS clocks for each matrix multiply!
- Compare this to the PS
  - ~1542 PS clocks per matrix multiply

# Where do I start with SDSoC?

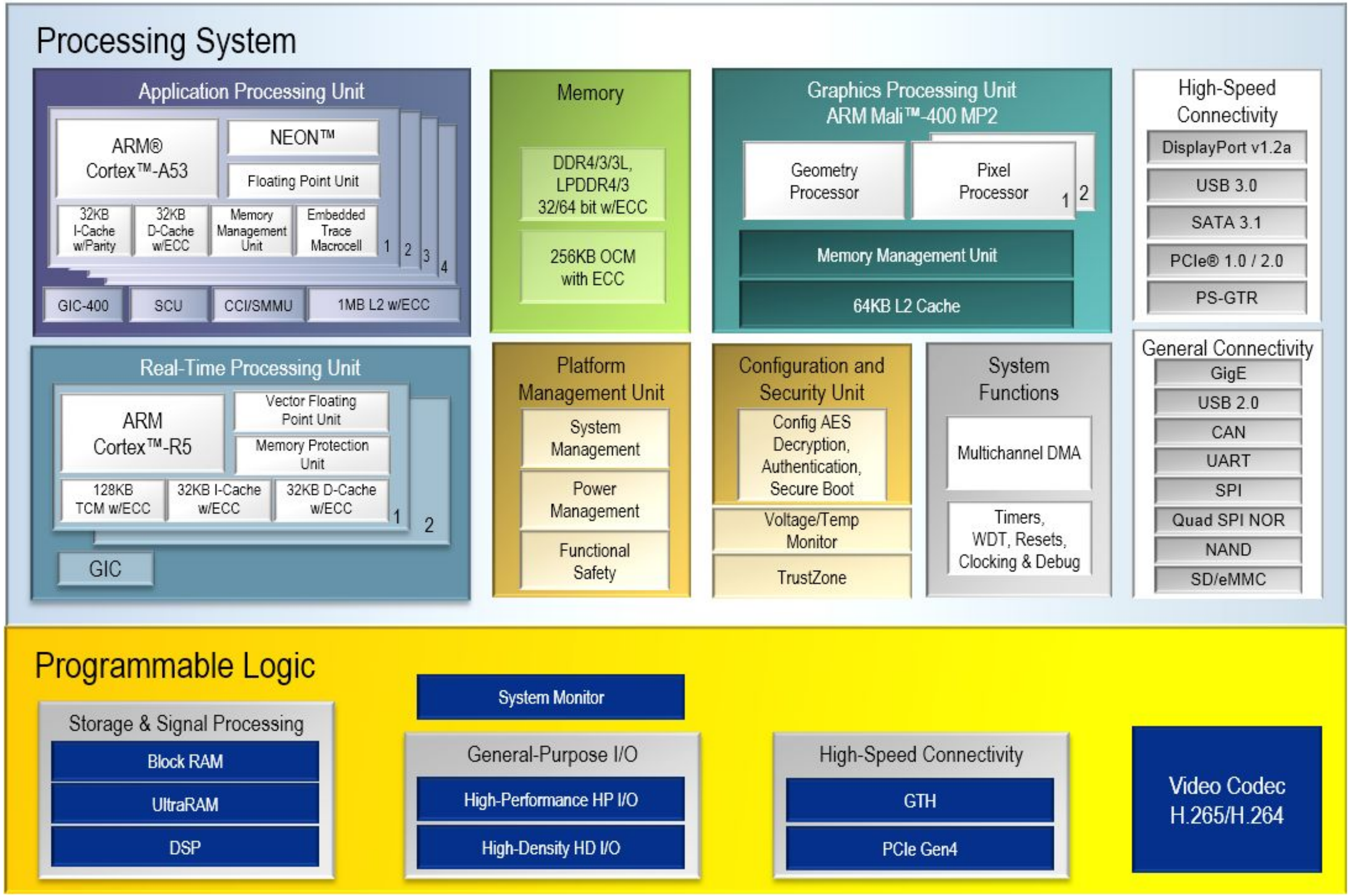
# What is Zynq?



| Cost-Optimized | Mid-Range | High-End | High-End |
|---|---|---|---|
| ZYNQ | ZYNQ<br>UltraSCALE+ | ZYNQ<br>UltraSCALE+ | ZYNQ<br>RFSoC |
| Zynq-7000 SoC<br>Artix Devices | Zynq UltraScale+ MPSoC<br>CG Devices | Zynq UltraScale+ MPSoC<br>EV Devices | Zynq UltraScale+ RFSoC<br>with RF Data Converters |
| Dual ARM® Cortex™-A9 | Dual ARM Cortex-A53<br>Dual ARM Cortex-R5 | Quad ARM Cortex-A53<br>Dual Cortex-R5 + GPU + Video Codec | Quad ARM Cortex-A53<br>Dual Cortex-R5 |
| Zynq-7000S SoC | ZYNQ | Zynq UltraScale+ MPSoC | Zynq UltraScale+ RFSoC |

Content Copyright Xilinx

# What is Zynq UltraScale+?



## Processing System

### Application Processing Unit

- ARM® Cortex™-A53
- NEON™
- Floating Point Unit
- 32KB I-Cache w/Parity
- 32KB D-Cache w/ECC
- Memory Management Unit
- Embedded Trace Macrocell
- 1 2 3 4
- GIC-400
- SCU
- CCI/SMMU
- 1MB L2 w/ECC

### Memory

- DDR4/3/3L, LPDDR4/3 32/64 bit w/ECC
- 256KB OCM with ECC

### Graphics Processing Unit ARM Mali™-400 MP2

- Geometry Processor
- Pixel Processor
- 1 2
- Memory Management Unit
- 64KB L2 Cache

### High-Speed Connectivity

- DisplayPort v1.2a
- USB 3.0
- SATA 3.1
- PCIe® 1.0 / 2.0
- PS-GTR

### Real-Time Processing Unit

- ARM Cortex™-R5
- Vector Floating Point Unit
- Memory Protection Unit
- 128KB TCM w/ECC
- 32KB I-Cache w/ECC
- 32KB D-Cache w/ECC
- 1 2
- GIC

### Platform Management Unit

- System Management
- Power Management
- Functional Safety

### Configuration and Security Unit

- Config AES Decryption, Authentication, Secure Boot
- Voltage/Temp Monitor
- TrustZone

### System Functions

- Multichannel DMA
- Timers, WDT, Resets, Clocking & Debug

### General Connectivity

- GigE
- USB 2.0
- CAN
- UART
- SPI
- Quad SPI NOR
- NAND
- SD/eMMC

## Programmable Logic

### Storage & Signal Processing

- Block RAM
- UltraRAM
- DSP

### System Monitor

### General-Purpose I/O

- High-Performance HP I/O
- High-Density HD I/O

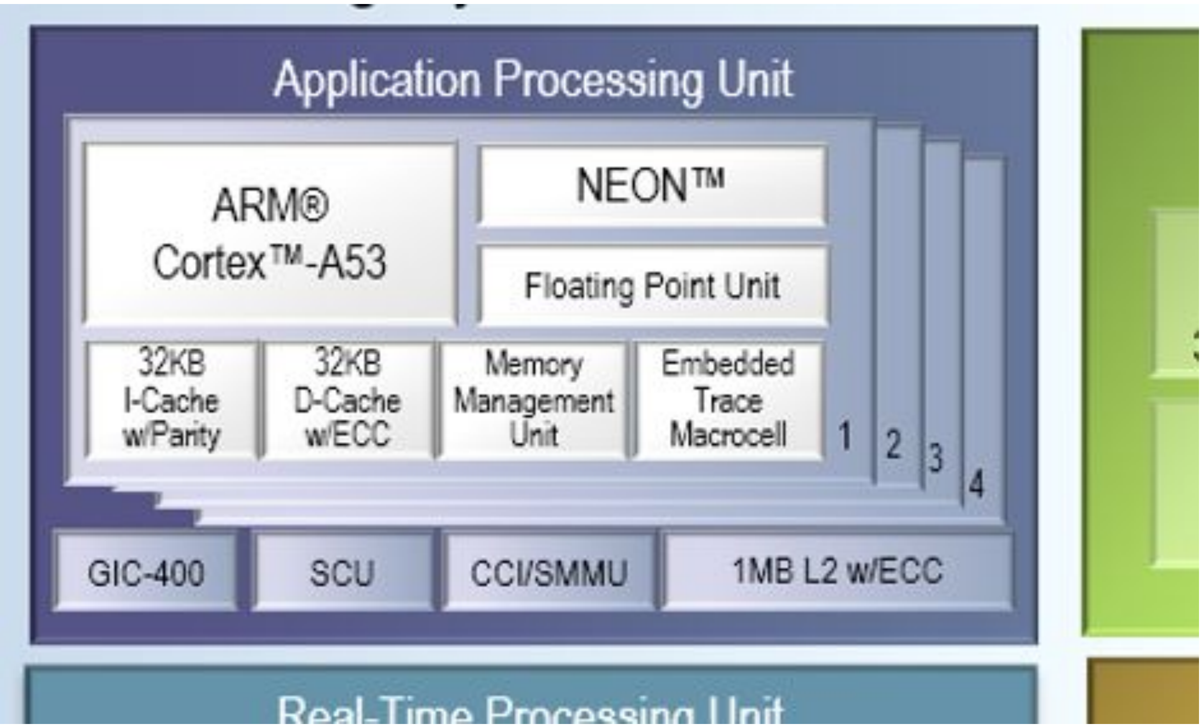
### High-Speed Connectivity

- GTH
- PCIe Gen4

### Video Codec H.265/H.264

Content Copyright Xilinx

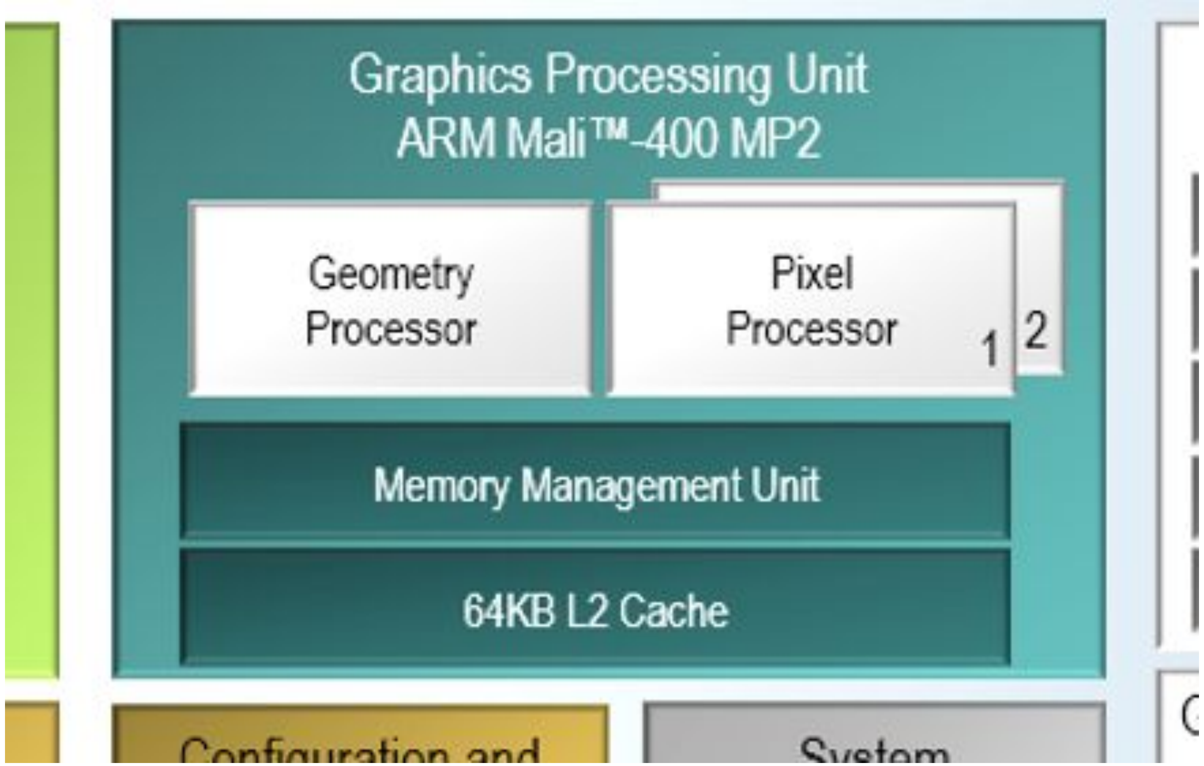
# What is Zynq UltraScale+?



- Quad Core Arm A53
- NEON SIMD Accelerators

Content Copyright Xilinx

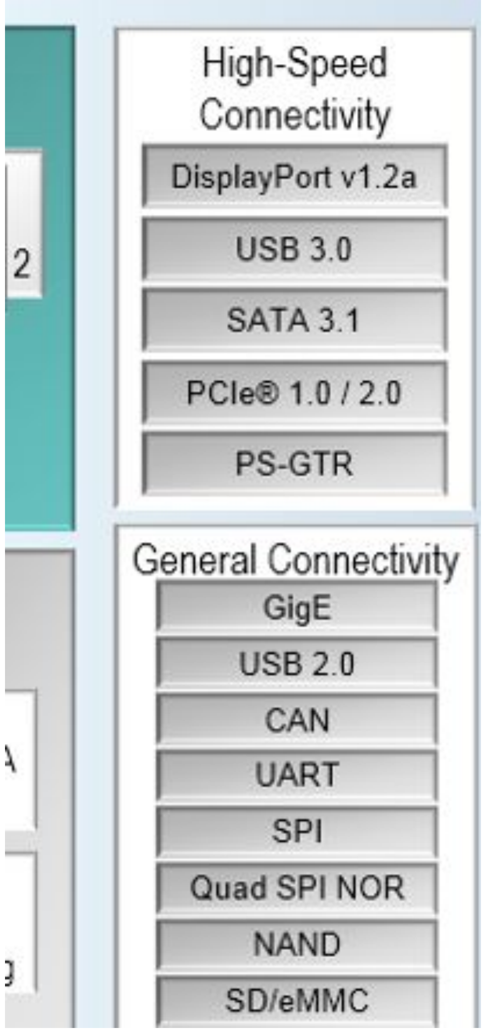
# What is Zynq UltraScale+?



- Quad Core Arm A53
- NEON SIMD Accelerators
- Mali GPU

Content Copyright Xilinx

# What is Zynq UltraScale+?



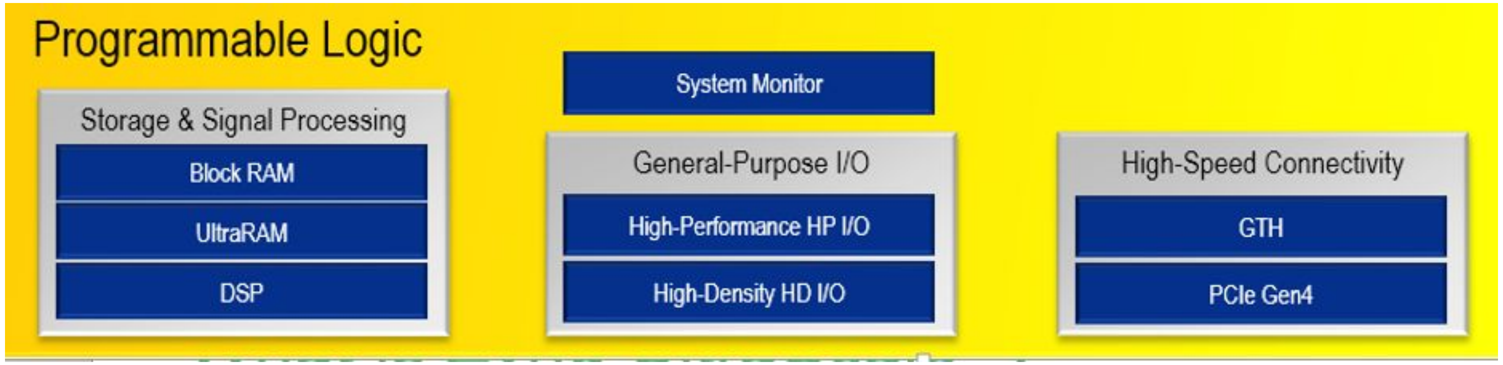
- Quad Core Arm A53
- NEON SIMD Accelerators
- Mali GPU
- Hardened IP
- VCU (Video Codec Unit)

Content Copyright Xilinx

# What is Zynq UltraScale+?

- Quad Core Arm A53
- NEON SIMD Accelerators
- Mali GPU
- Hardened IP
- VCU (Video Codec Unit)

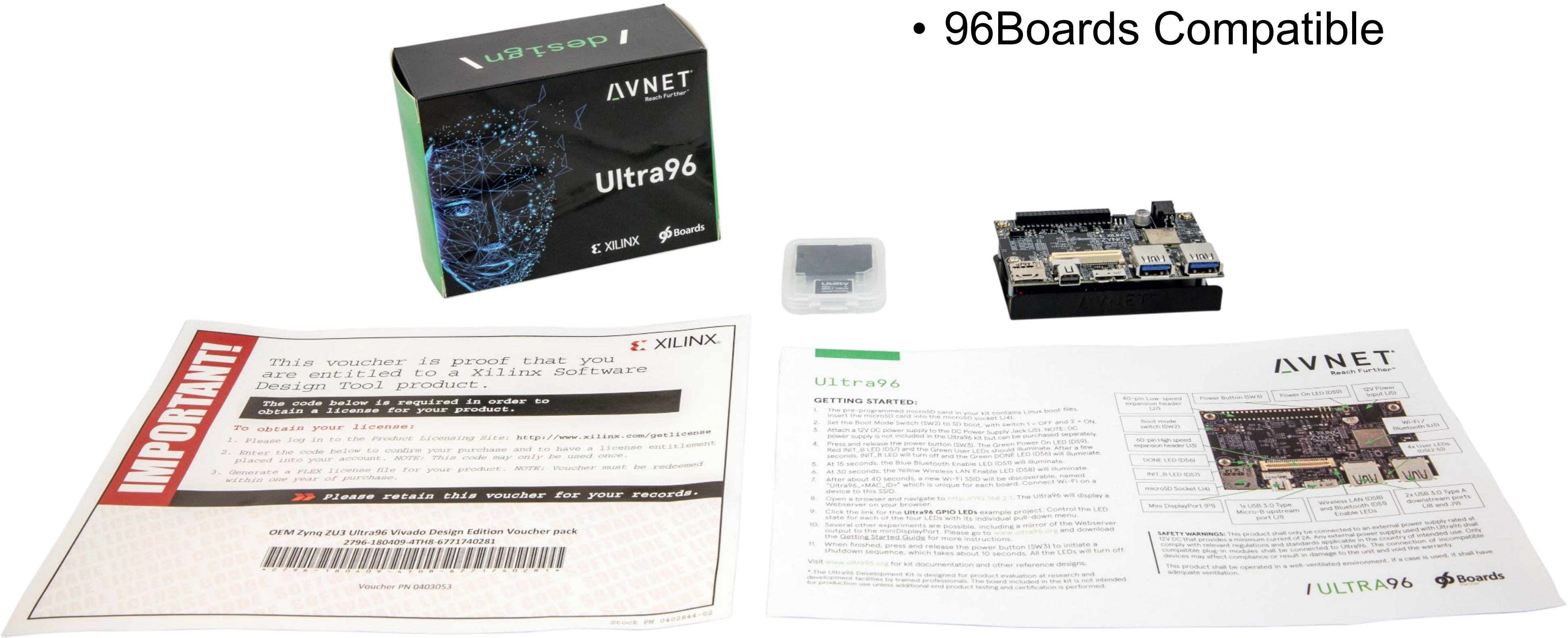
- Variable amounts of Programmable Logic (amount of acceleration possible)
- Variable grades of Programmable Logic (speed)
- Variable fabric families of Programmable Logic (performance)



Content Copyright Xilinx

# What is Zynq UltraScale+?

- EX: Ultra96 $249; comes with SDSoC License
- 96Boards Compatible



This voucher is proof that you are entitled to a Xilinx Software Design Tool product.

**The code below is required in order to obtain a license for your product.**

*To obtain your license:*

1. *Please log in to the Product Licensing Site:* http://www.xilinx.com/getlicense
2. *Enter the code below to confirm your purchase and to have a license entitlement placed into your account. NOTE: This code may only be used once.*
3. *Generate a FLEX license file for your product. NOTE: Voucher must be redeemed within one year of purchase.*

***Please retain this voucher for your records.***

*OEM Zynq ZU3 Ultra96 Vivado Design Edition Voucher pack*

*2796-180409-4TH8-6771740281*

*2796-180409-4TH8-6771740281*

*Voucher PN 0403053*

Stock PN 0402844-02

Ultra96

**GETTING STARTED:**

1. The pre-programmed microSD card in your kit contains Linux boot files. Insert the microSD card into the microSD socket (J4).
2. Set the Boot Mode Switch (SW2) to SD boot, with switch 1 = OFF and 2 = ON.
3. Attach a 12V DC power supply to the DC Power Supply Jack (J5). NOTE: DC power supply is not included in the Ultra96 kit but can be purchased separately.
4. Press and release the power button (SW3). The Green Power On LED (DS9), Red INIT_B LED (DS7) and the Green User LEDs should illuminate. After a few seconds, INIT_B LED will turn off and the Green DONE LED (DS6) will illuminate.
5. At 15 seconds, the Blue Bluetooth Enable LED (DS1) will illuminate.
6. At 30 seconds, the Yellow Wireless LAN Enable LED (DS8) will illuminate.
7. After about 40 seconds, a new Wi-Fi SSID will be discoverable, named "Ultra96_<MAC_ID>" which is unique for each board. Connect Wi-Fi on a device to this SSID.
8. Open a browser and navigate to http://192.168.2.1. The Ultra96 will display a Webserver on your browser.
9. Click the link for the **Ultra96 GPIO LEDs** example project. Control the LED state for each of the four LEDs with its individual pull-down menu.
10. Several other experiments are possible, including a mirror of the Webserver output to the miniDisplayPort. Please go to www.ultra96.org and download the Getting Started Guide for more instructions.
11. When finished, press and release the power button (SW3) to initiate a shutdown sequence, which takes about 10 seconds. All the LEDs will turn off.

Visit www.ultra96.org for kit documentation and other reference designs.

* The Ultra96 Development Kit is designed for product evaluation at research and development facilities by trained professionals. The board included in the kit is not intended for production use unless additional end product testing and certification is performed.

AVNET
Reach Further
40-pin Low-speed expansion header (J7)
Power Button (SW3)
Power On LED (DS9)
12V Power Input (J5)
Boot mode switch (SW2)
Wi-Fi / Bluetooth (U3)
60-pin High speed expansion header (J3)
4x User LEDs (DS[2:5])
DONE LED (DS6)
INIT_B LED (DS7)
microSD Socket (J4)
Mini DisplayPort (P1)
1x USB 3.0 Type Micro-B upstream port (J1)
Wireless LAN (DS8) and Bluetooth (DS1) Enable LEDs
2x USB 3.0 Type A downstream ports (J8 and J9)

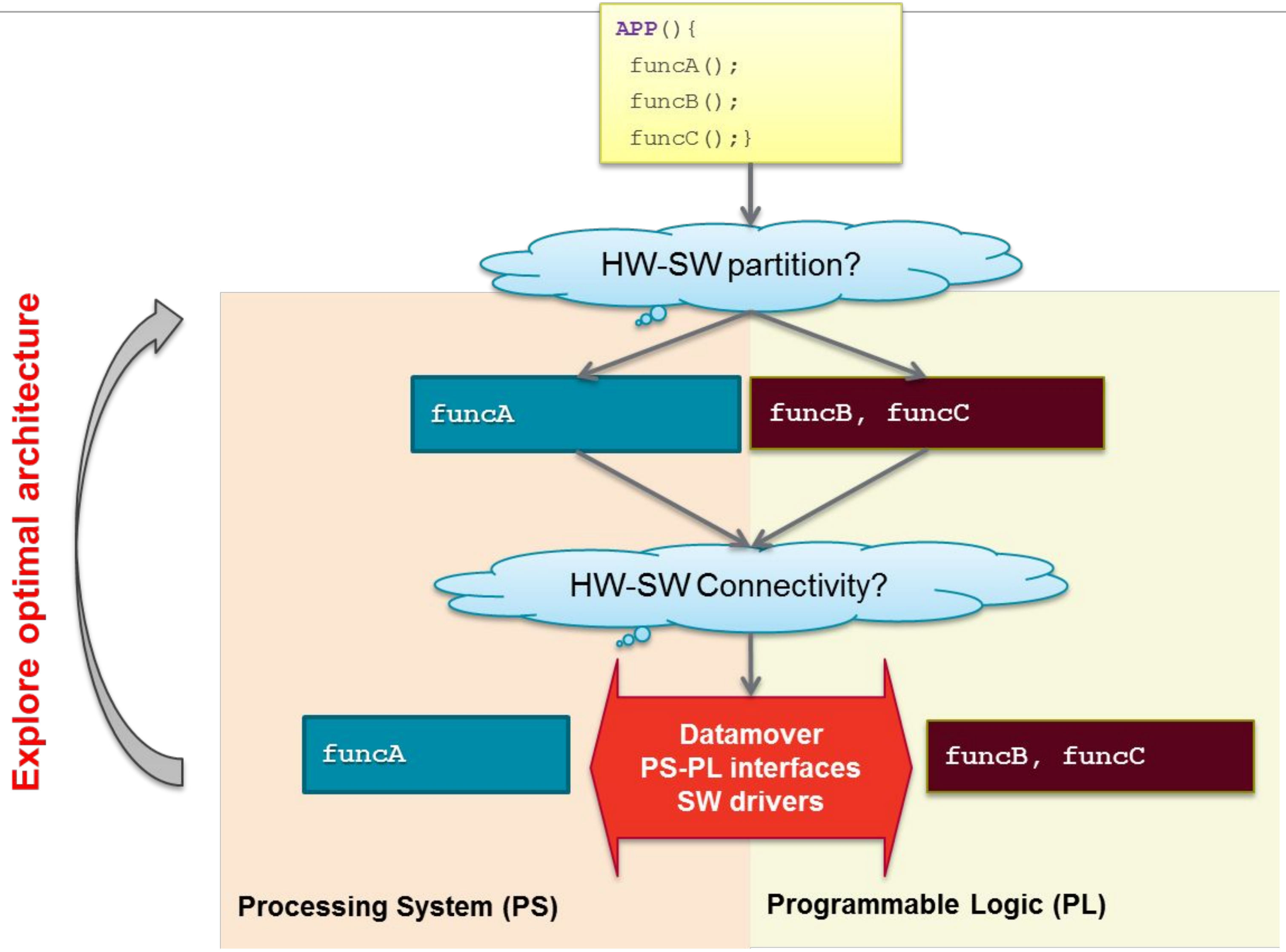
**SAFETY WARNINGS:** This product shall only be connected to an external power supply rated at 12V DC that provides a minimum current of 2A. Any external power supply used with Ultra96 shall comply with relevant regulations and standards applicable in the country of intended use. Only compatible plug-in modules shall be connected to Ultra96. The connection of incompatible devices may affect compliance or result in damage to the unit and void the warranty.

This product shall be operated in a well-ventilated environment. If a case is used, it shall have adequate ventilation.

Content Copyright Xilinx

# What is my design flow?

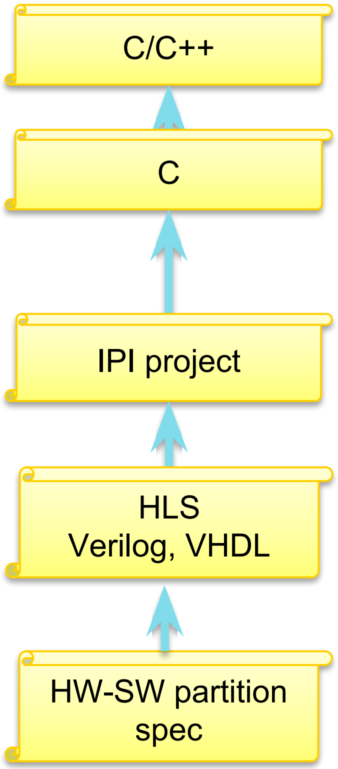
# Typical Zynq Development Flow



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

Content Copyright Xilinx

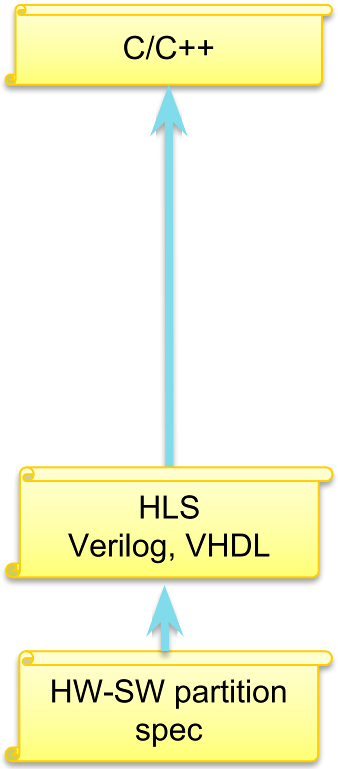
# Before SDSoC:



- Involves Multiple Disciplines
- Direct manual coordination for solution

Content Copyright Xilinx

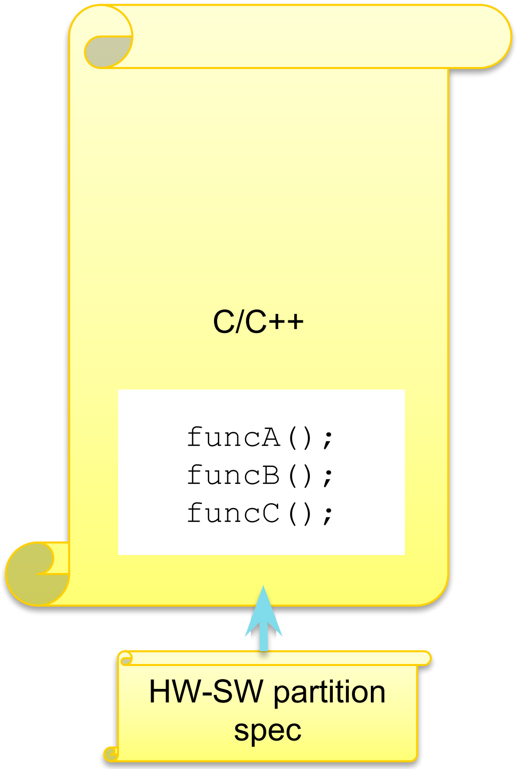
# After SDSoC:



- Remove the manual design of SW drivers and HW connectivity

Content Copyright Xilinx

# After SDSoC:

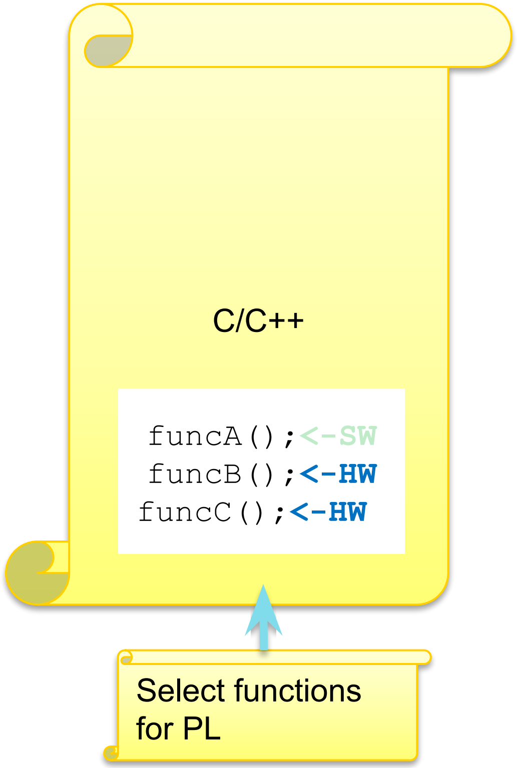


```
funcA();
funcB();
funcC();
```

- 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

Content Copyright Xilinx

# After SDSoC:



C/C++

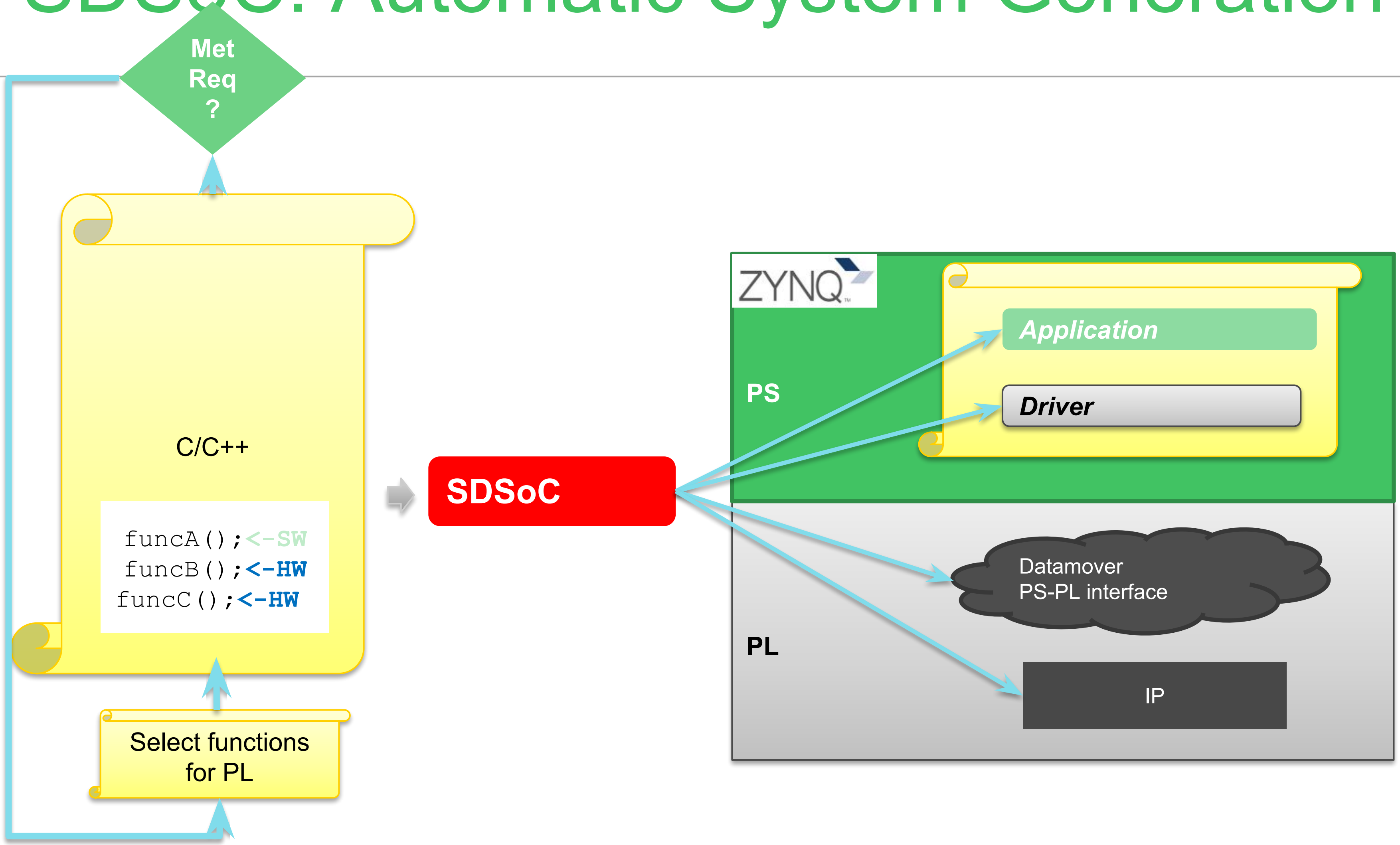
```
 funcA();<-SW
 funcB();<-HW
funcC();<-HW
```

Select functions for PL

- 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

Content Copyright Xilinx

# After SDSoC: Automatic System Generation



C/C++ to System in hours, not days

Content Copyright Xilinx

# Platforms… what are they?

# Platforms…what are they?

- A *platform* is a base system designed for reuse, our sandbox
  - Built using Vivado, SDK (now optional) and OS tools
  - Your Hardware is Defined: Processing system, I/O subsystems, memory interfaces, …, with a well-defined port interface (AXI, AXI-S, clock, reset, interrupt)
  - Software Definition: OS, device drivers, boot loaders, file system, libraries,…

# Platforms…what are they?

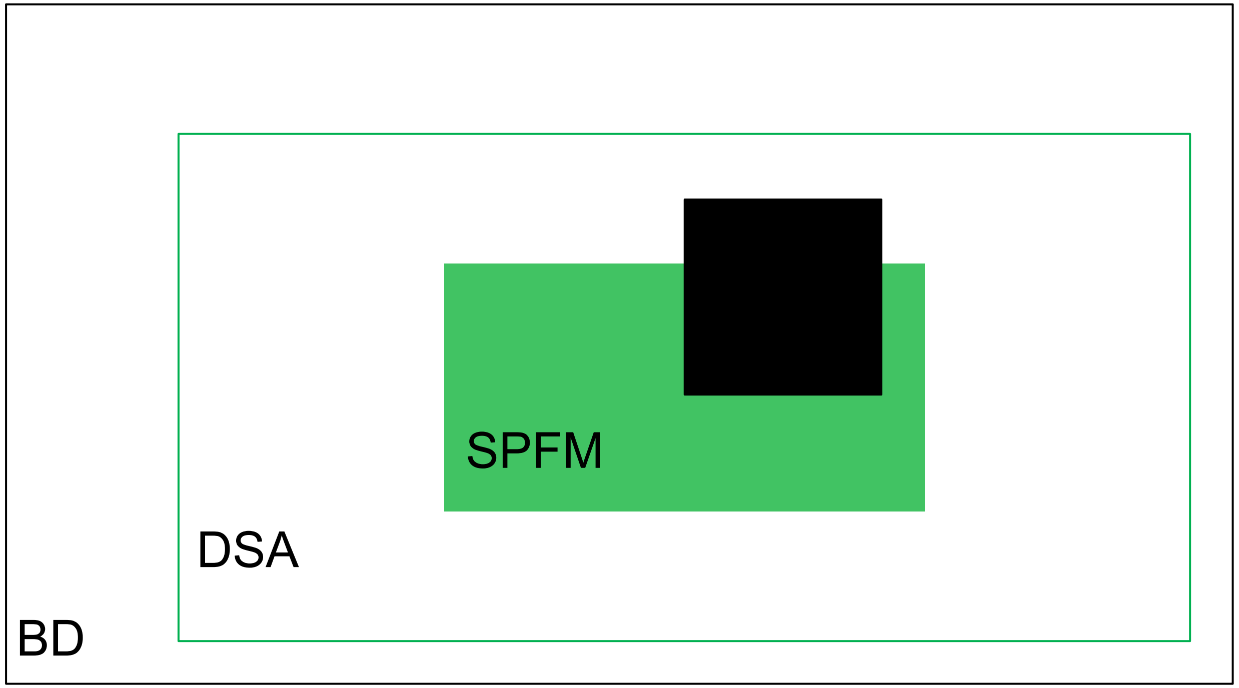
- SDSoC extends the platform with application-specific hardware and software
  - Complete solution generated from C/C++ source
  - Solution can include RTL packaged

- SDSoC treats platforms as independent solution spaces
  - Generate IPs for your solution based on the resources available
  - Each solution is tailored for each platform
  - A solution can have the target platform switched through little more than a few GUI clicks!

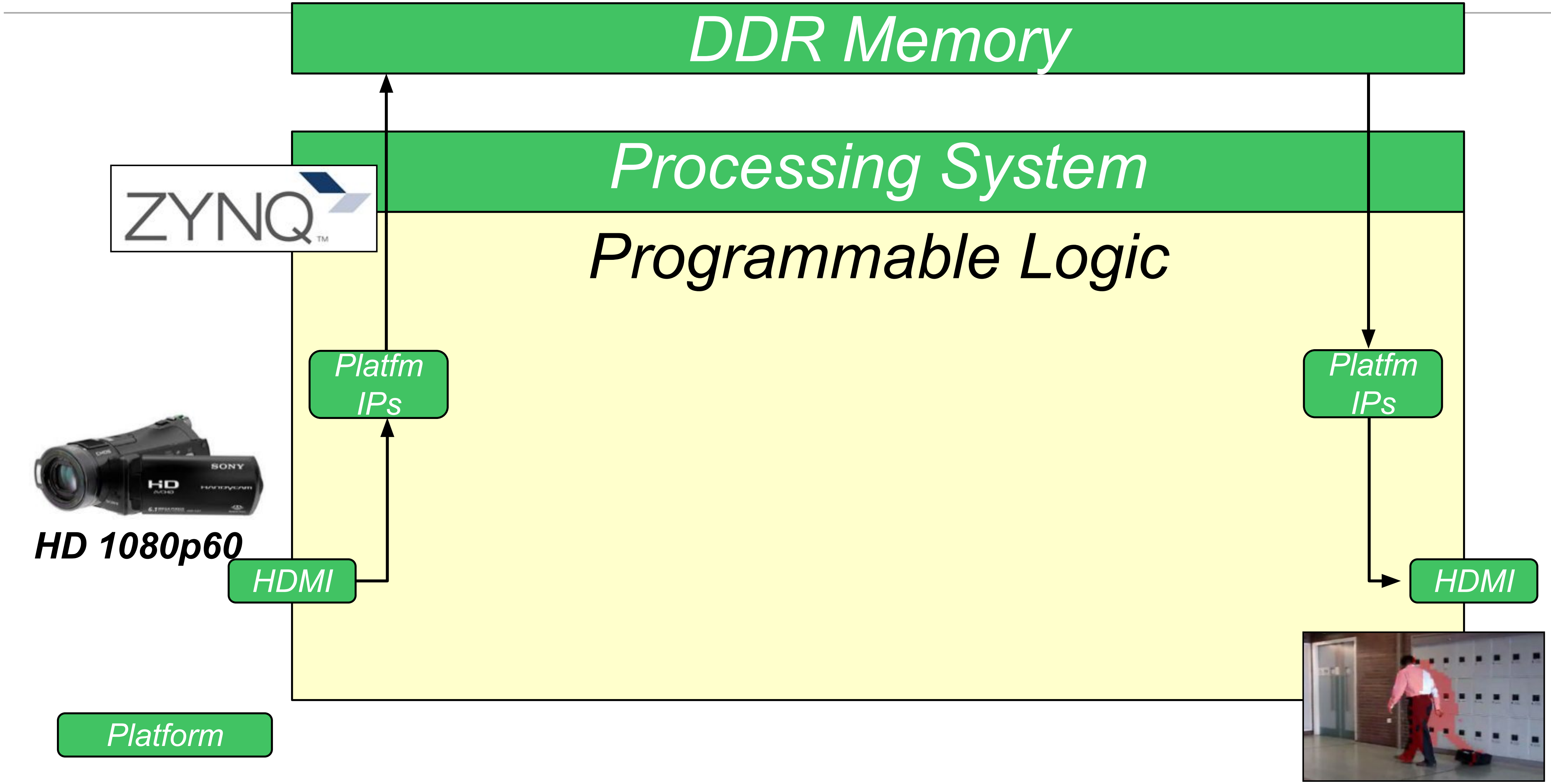
- Platforms must include
  - Processing IP Block
  - At least one general purpose AXI master port

# XPFM: A Pictoral

- XPFM
  - Platform Reference to the location of the DSA and SPFM
- BD
  - Block Design
- DSA
  - Device Support Archive
- SPFM
  - Software Platform Descriptor XML
- Black Box
  - SDSoC created IP



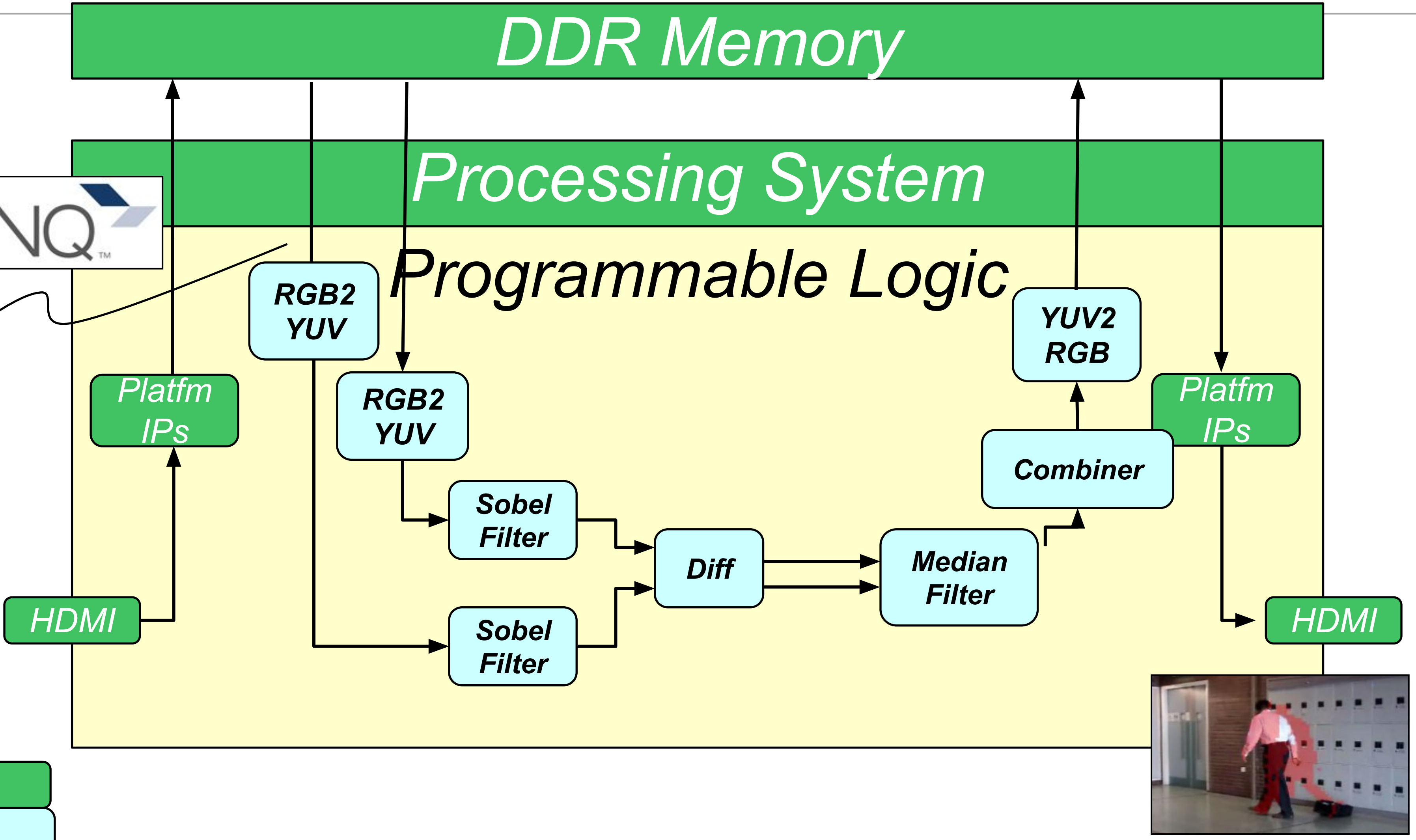
# Example: Optical Flow Design



Content Copyright Xilinx

# Example: Motion Detection Application

Data Movers added and managed



HD 1080p60

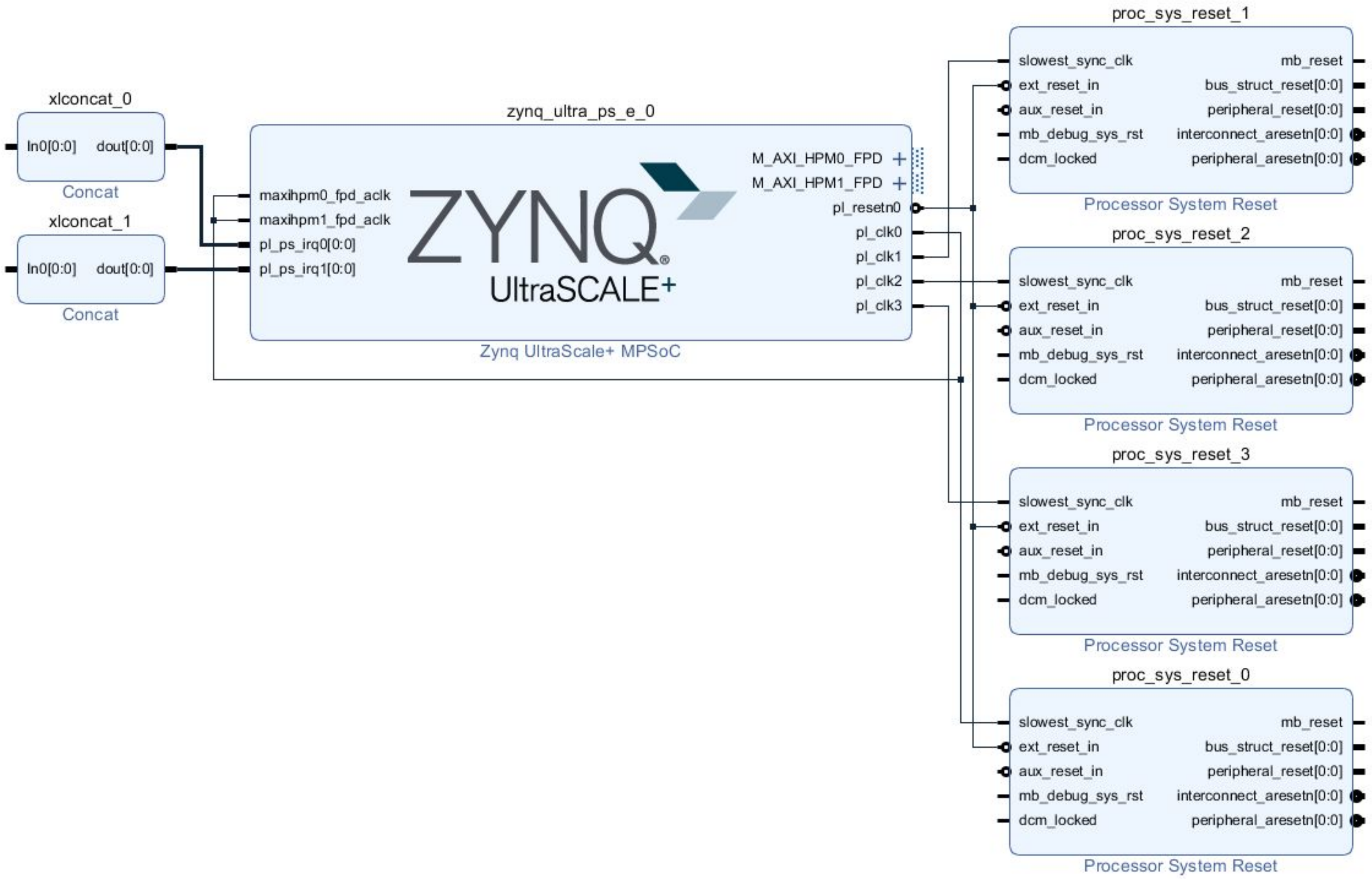
Platform

Image Processing

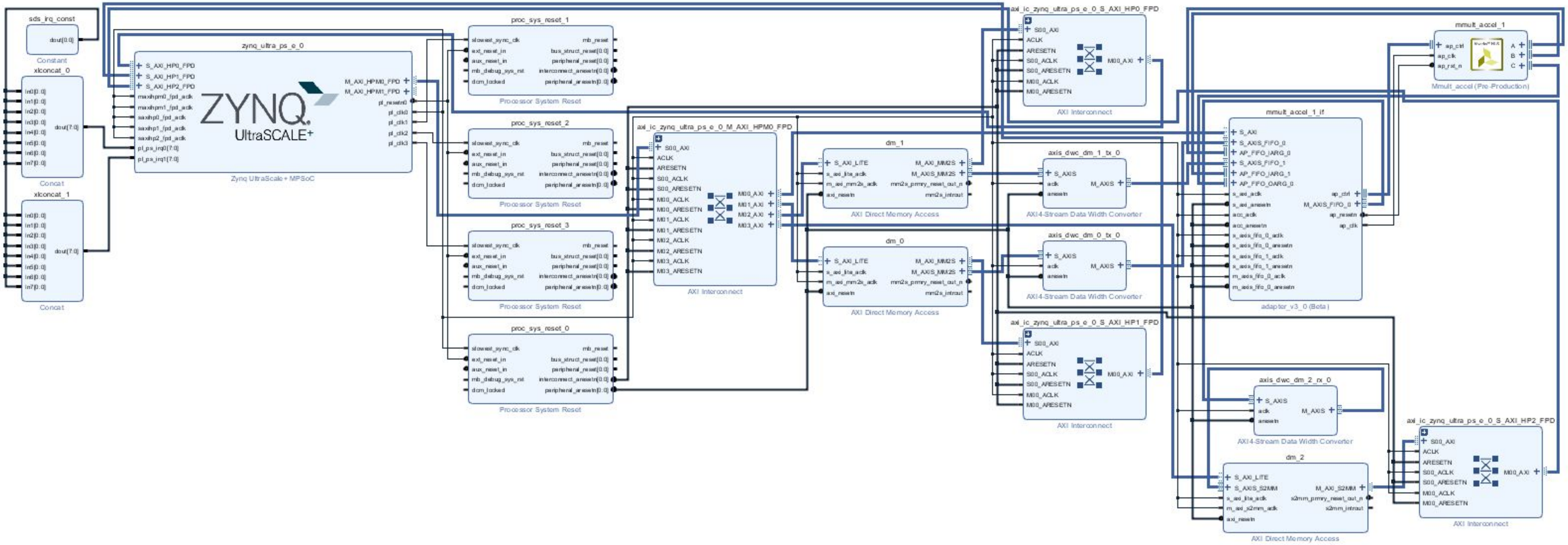
Content Copyright Xilinx

AVNET

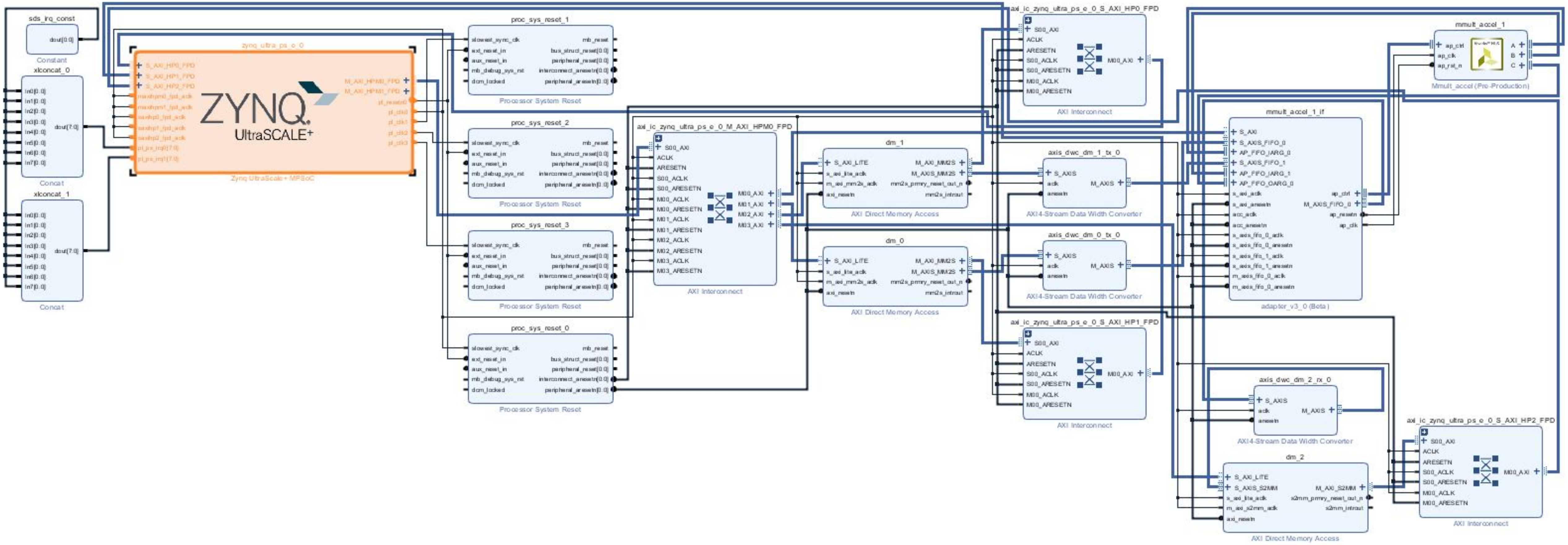
# Example: Ultra96 Platform



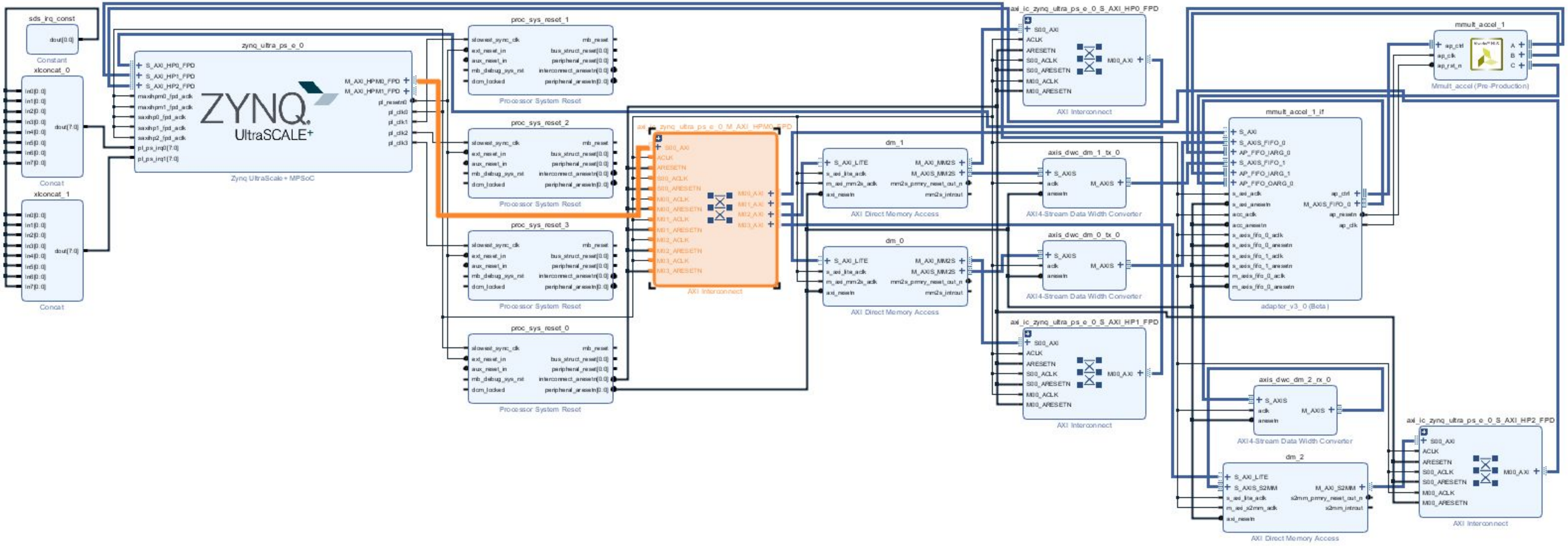
# Example: Ultra96 Platform with mmult



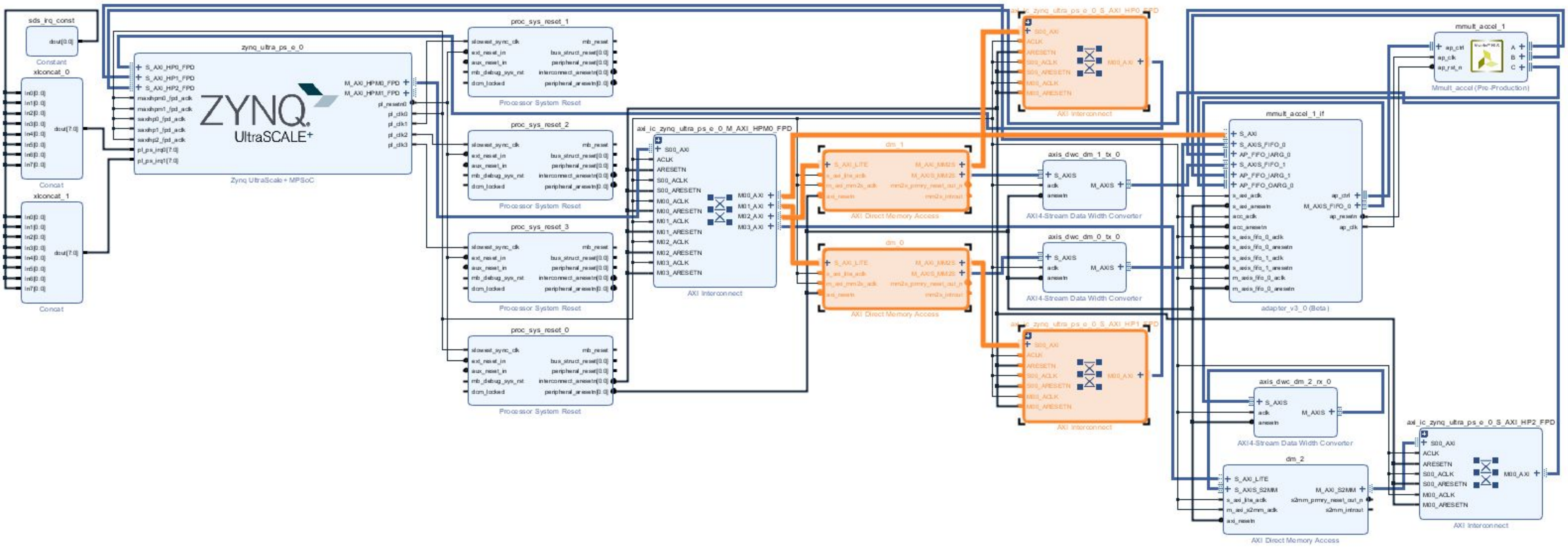
# Example: Ultra96 Platform with mmult



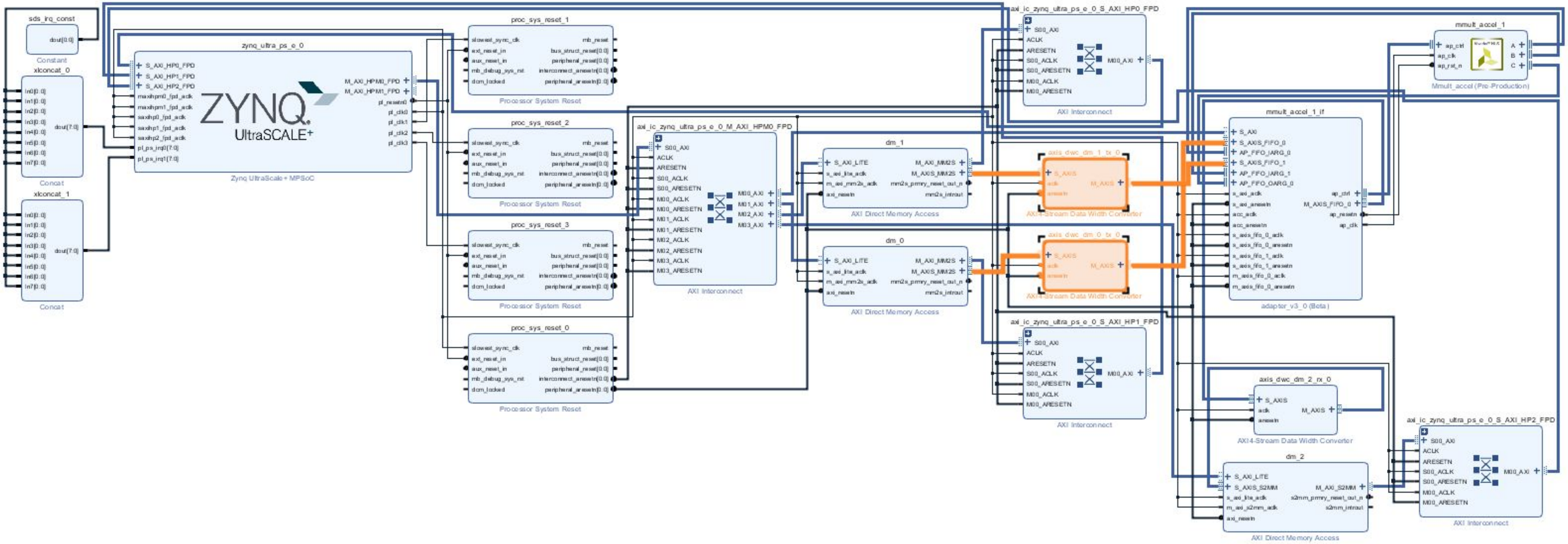
# Example: Ultra96 Platform with mmult



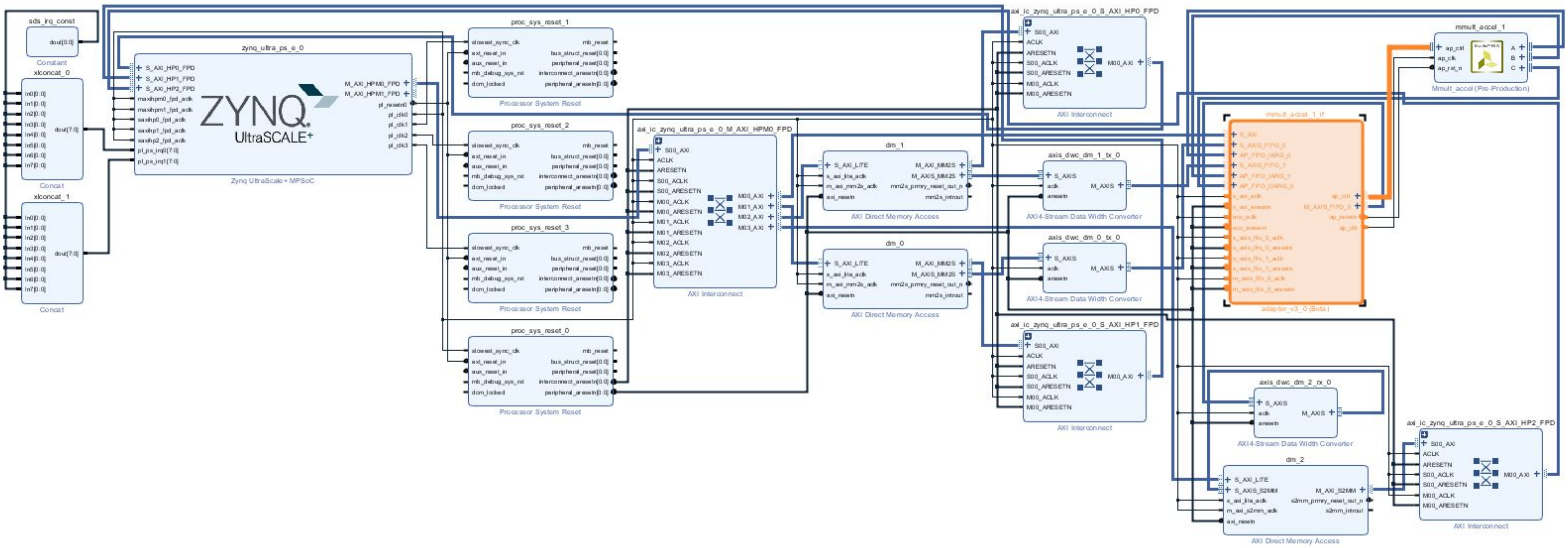
# Example: Ultra96 Platform with mmult



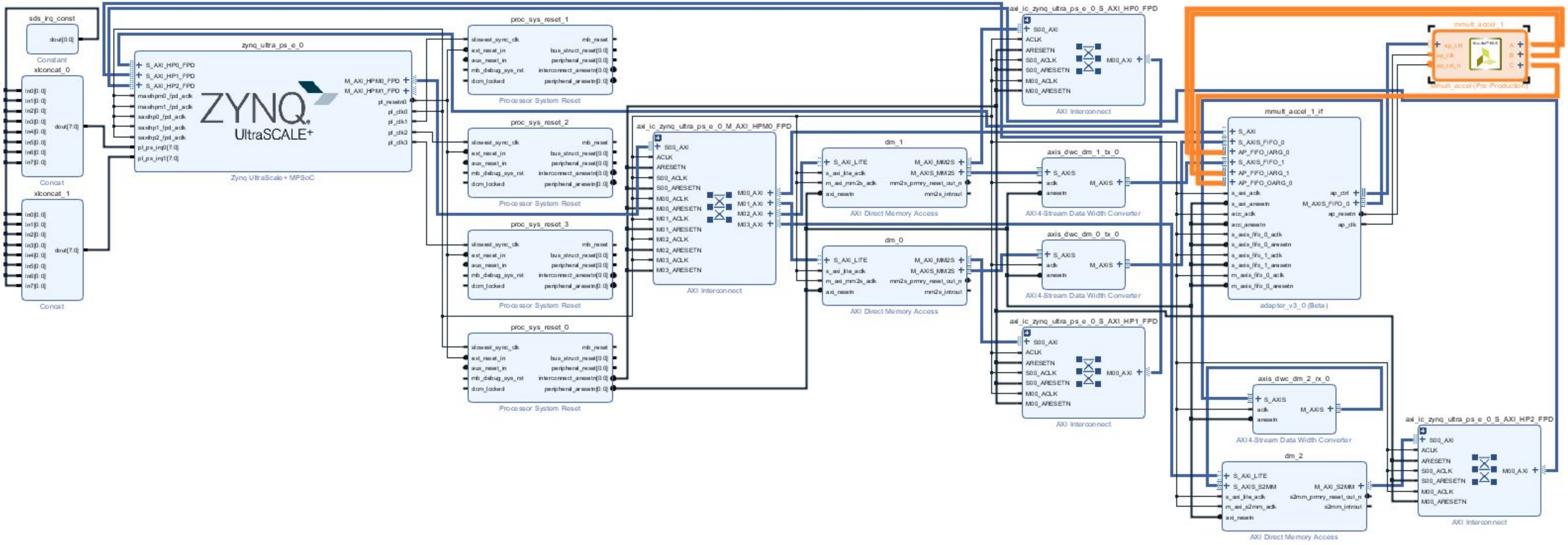
# Example: Ultra96 Platform with mmult



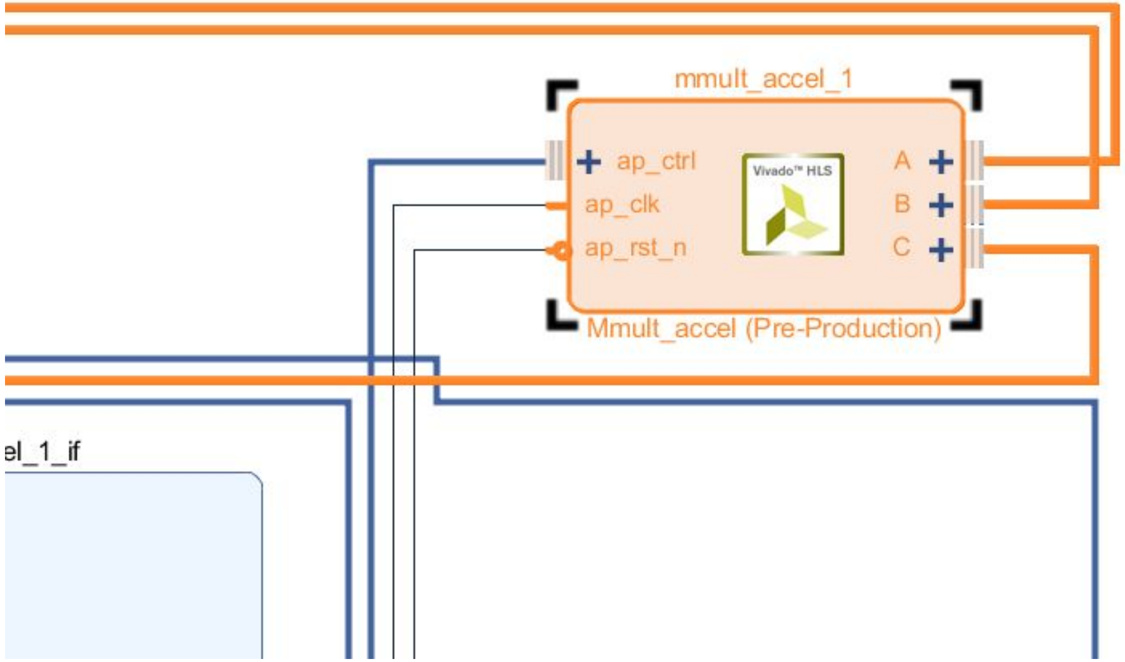
# Example: Ultra96 Platform with mmult



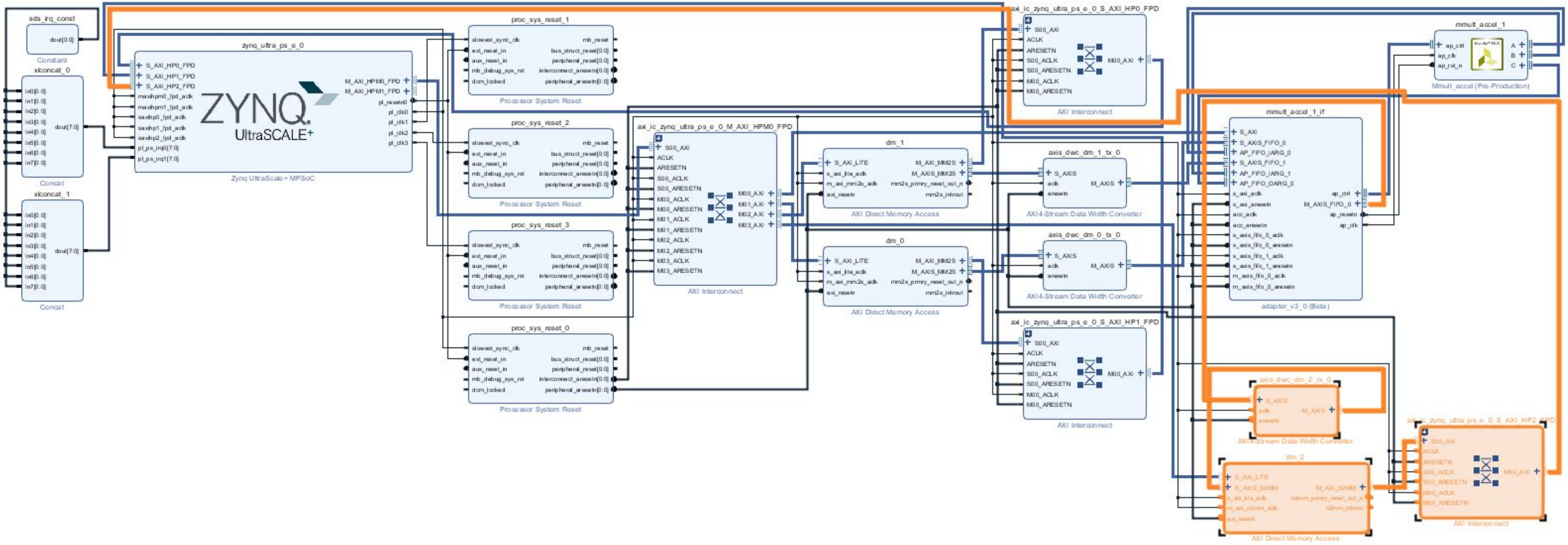
# Example: Ultra96 Platform with mmult



# Example: Ultra96 Platform with mmult



# Example: Ultra96 Platform with mmult



AVNET

# Thank you!

# Features

# C-Callable IP

C-Callable IP

- Allows the reuse legacy IP and provide a software-centric handoff from hardware / IP team to application developers
- Launch from SDx Terminal

Some Content Copyright Xilinx

# Additional Resources

# For more Information on HLS or Pragmas

SDx Developer Topics

- https://www.xilinx.com/html_docs/xilinx2017_2/sdaccel_doc/topics/pragmas/ref-pragma_HLS_interface.html

Getting Started with Vivado High-Level Synthesis

- https://www.xilinx.com/video/hardware/getting-started-vivado-high-level-synthesis.html

- Avnet's SDSoC Advanced Concepts – Streaming IO Example
  - Please visit the minized.org website or contact your local Avnet FAE for access
  - This follows the UG1146 Example injecting streaming IO control into an SDSoC Platform
  - Based on our MiniZed Platform

# For more Information on HLS or Pragmas

Xilinx Documentation (not all inclusive)

- Vivado Design Suite Tutorial High-Level Synthesis UG871 (v2017.4)
- Vivado Design Suite User Guide High-Level Synthesis UG902 (v2017.4)
- SDSoC Environment User Guide UG1027 (v2017.4)
- SDx Pragma Reference Guide UG1253 (v2017.4)

# Basic HLS

# HLS

High Level Synthesis

- Vivado High-Level Synthesis compiler
  - C, C++ and SystemC programs directly targeted into Xilinx devices
  - Without the need to manually create RTL

https://en.wikipedia.org/wiki/Xilinx_Vivado

- What does this mean?
  - You can write HLS compliant C/C++ code and the TOOL will translate that to RTL!

# How does it know what to do?

SDSoC Uses

- Modeling
- Estimation
- Predictive Analysis

Through this, SDSoC can

- Utilize all available resources to create the most optimal design given the resources available

# How can I tell the tool what I want?

- SDSoC Leverages a Suite of Tools, one being Vivado HLS pragmas!
- In C based design, all input and output operations are performed, in zero time, through formal function arguments.
- In an RTL design these same input and output operations must be performed through a port in the design interface and typically operate using a specific I/O (input-output) protocol.
- For more information, refer to "Managing Interfaces" in the Vivado Design Suite User Guide: High-Level Synthesis (UG902).

# How can I tell the tool what I want?

In other words

- Pragmas are comments placed into the C/C++ code that instructs the C->RTL conversion to have specific bounds and goals

- Remember, SDSoC sits on TOP of a suite of tools
  - While there are HLS specific pragmas, there are also SDS pragmas to be used with the SDx family

# Pragmas Use In Our Example

mmult.h

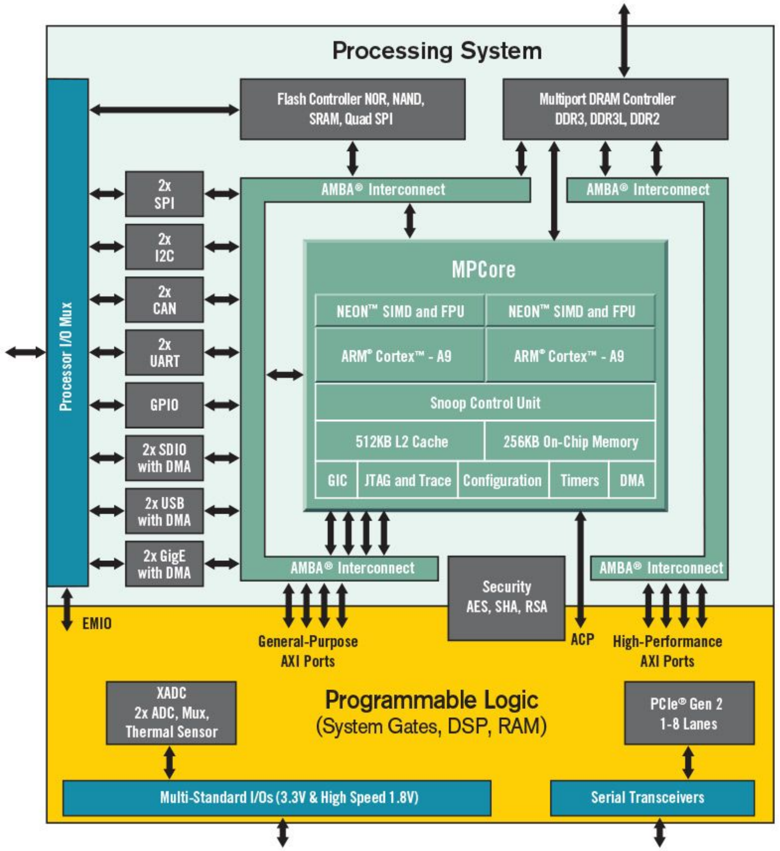
- #pragma SDS data access_pattern
  - This pragma specifies the data access pattern in the hardware function.
  - By describing the KNOWN data access pattern, SDSoC can best determine what data movers should be used

mmult.cpp

- #pragma HLS array_partition
  - Partitions an array into smaller arrays or individual elements.
  - In this case used to spread out array allowing parallel access to all elements
- #pragma HLS PIPELINE
  - The PIPELINE pragma reduces the initiation interval for a function or loop by allowing the concurrent execution of operations.
  - Adding this will generate a pipelined feeder to ensure new data is available to the function or look every N clock cycles
  - In this case every 1 clock cycle

*information from Xilinx SDx Developer Topics website

# What is Zynq?



- Single or Dual Core Arm A9
- NEON SIMD Accelerators
- Variable amounts of Programmable Logic (amount of acceleration possible)
- Variable grades of Programmable Logic (speed)
- Variable fabric families of Programmable Logic (performance)

Content Copyright Xilinx

# What is Zynq?

- EX: Avnet MiniZed $88.99; comes with SDSoC License
- Arduino Compatible

Content Copyright Xilinx