



Getting started with FPGA

Rinku Shah¹, Assistant Professor, IIIT Delhi

Neeraj Kumar Yadav¹, Research Scholar, IIIT Delhi

Keshav Gambhir², Microsoft India

Harish S A³, Research Scholar (PMRF), IIT Hyderabad



¹ IIIT Delhi
India

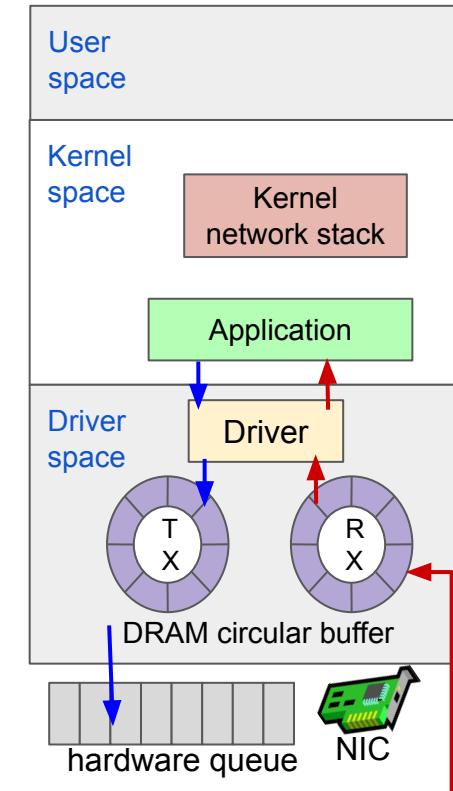
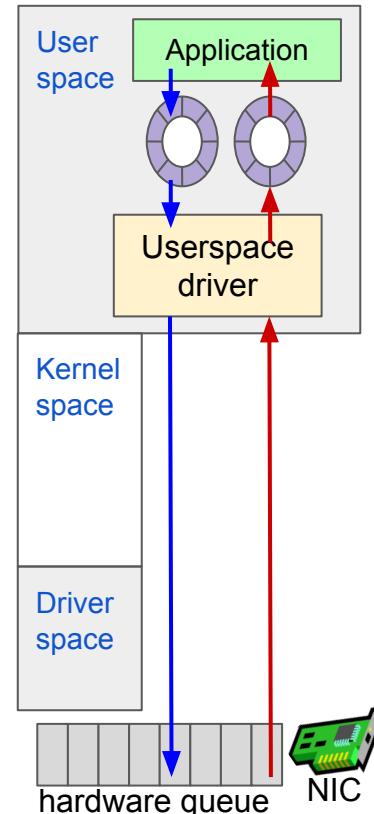
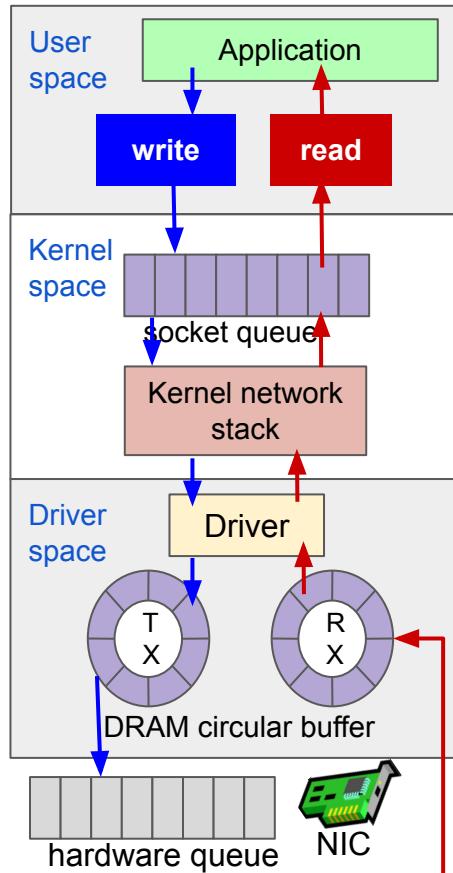


² Microsoft
India



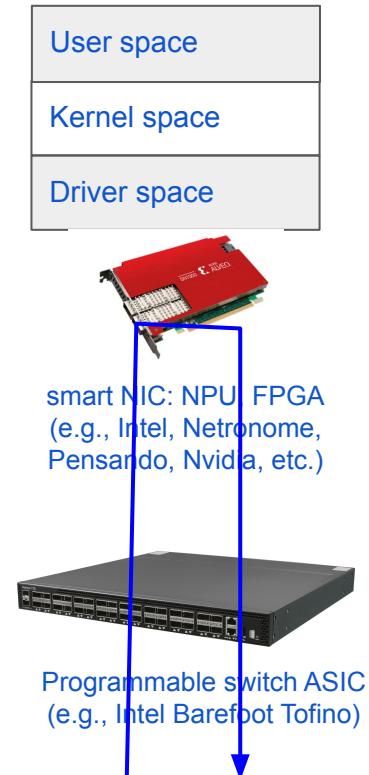
³ IIT Hyderabad
India

Evolution of network packet processors



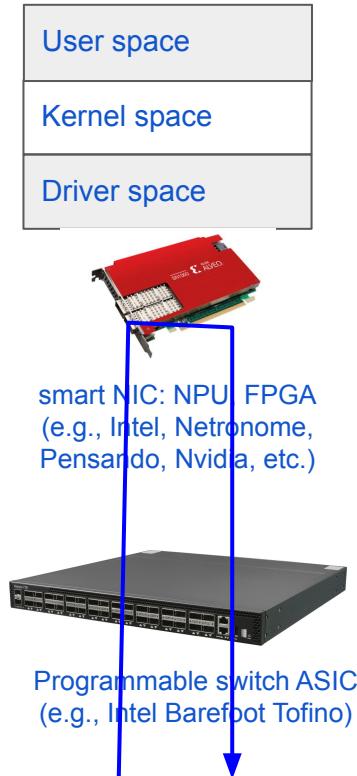
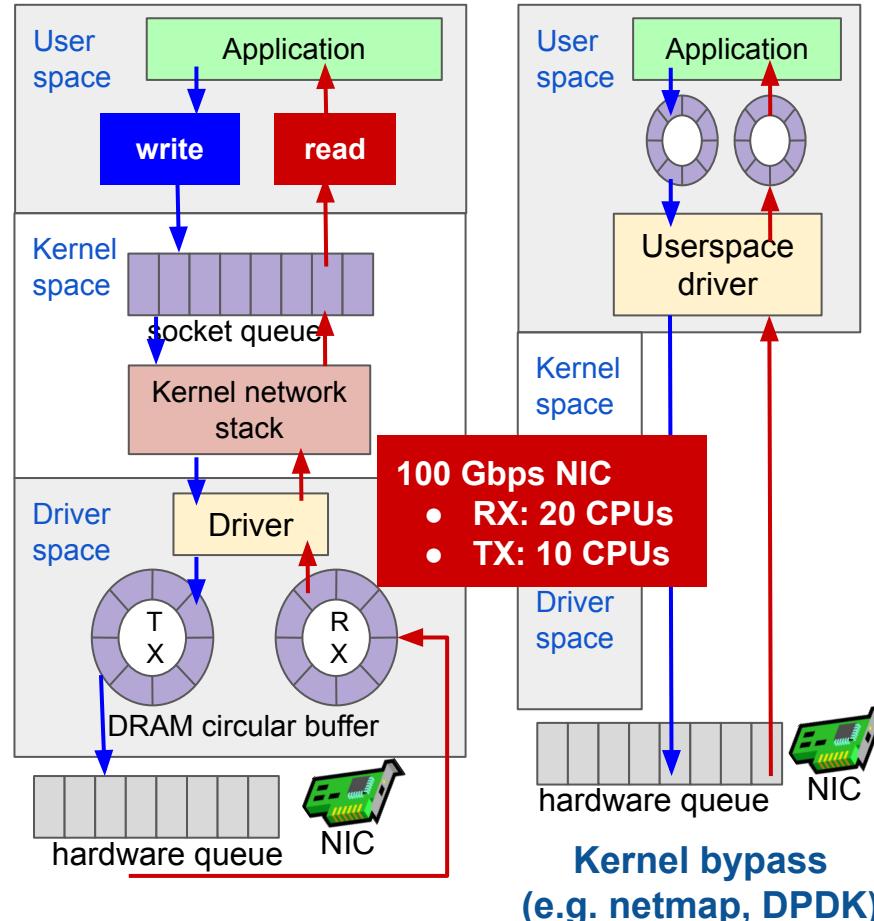
Kernel bypass
(e.g. netmap, DPDK)

in-kernel compute
(e.g. eBPF, XDP)



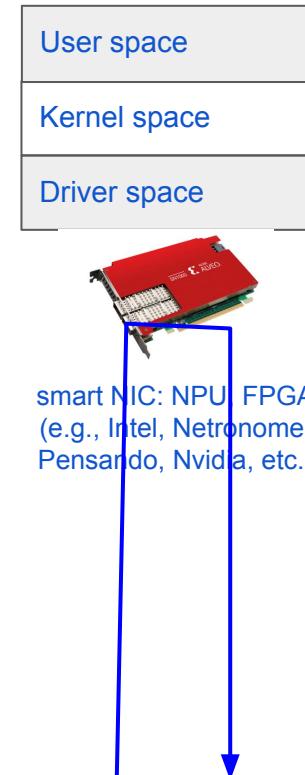
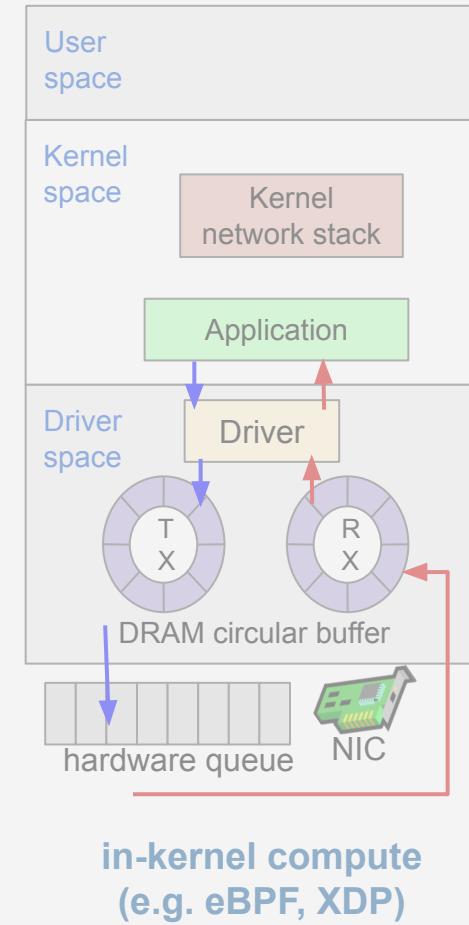
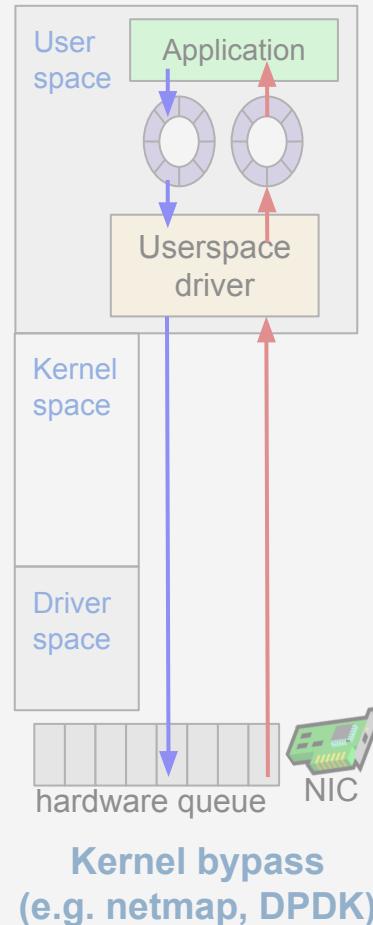
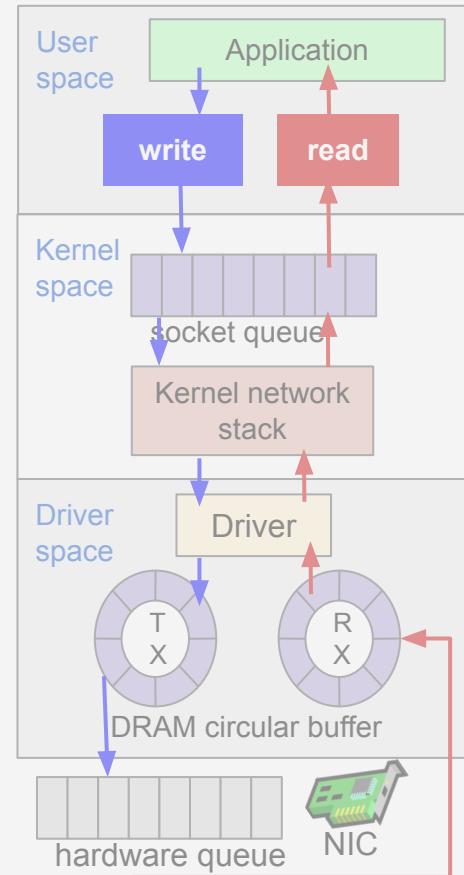
in-network compute
(NIC/switch)

Evolution of network packet processors

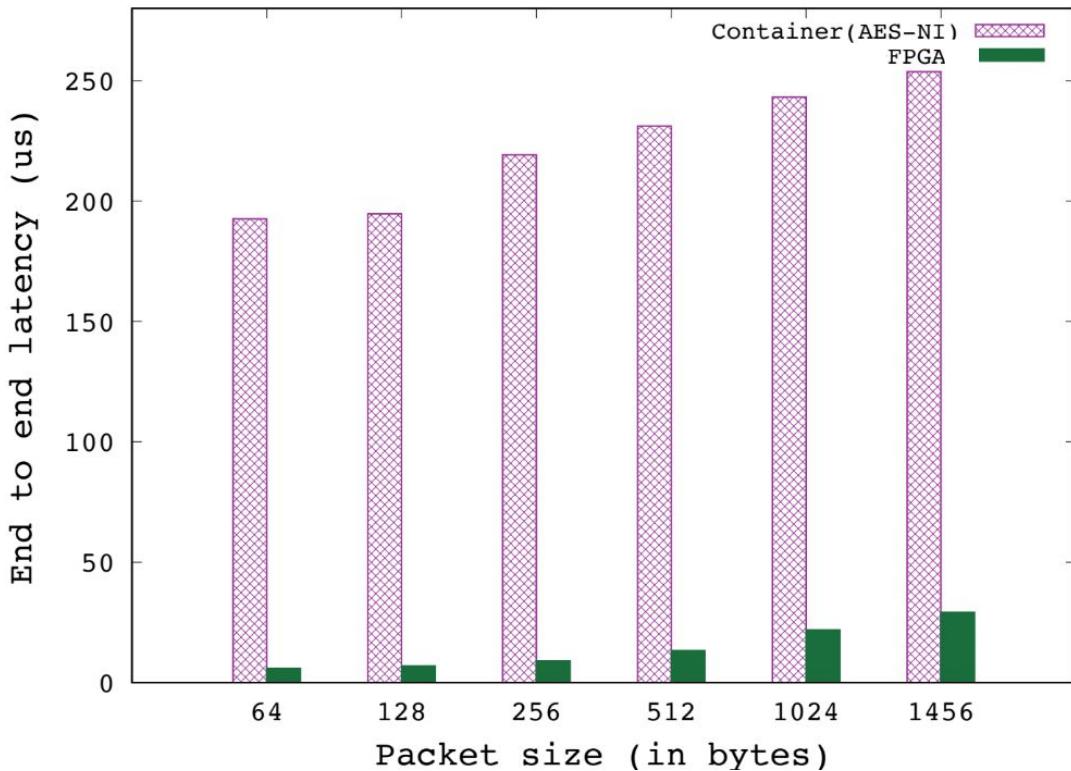


**in-network compute
(NIC/switch)**

Evolution of network packet processors

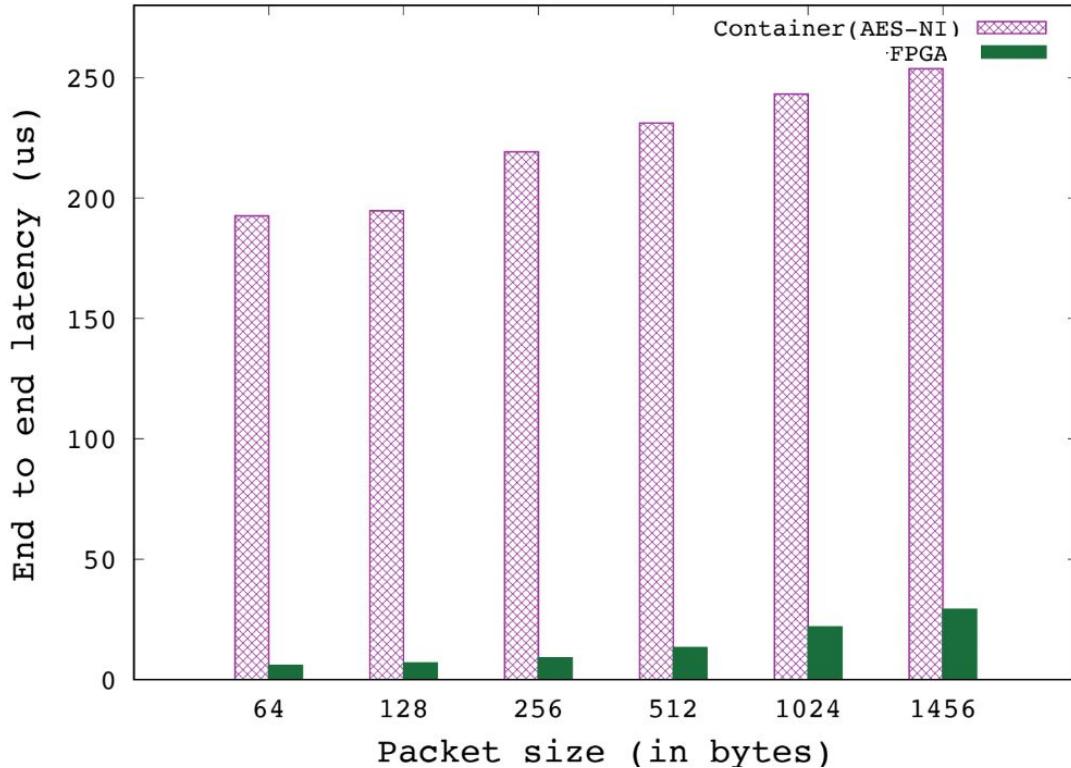


Offloading application to FPGA NIC



AES-GCM: Cryptography algorithm used in TLS.

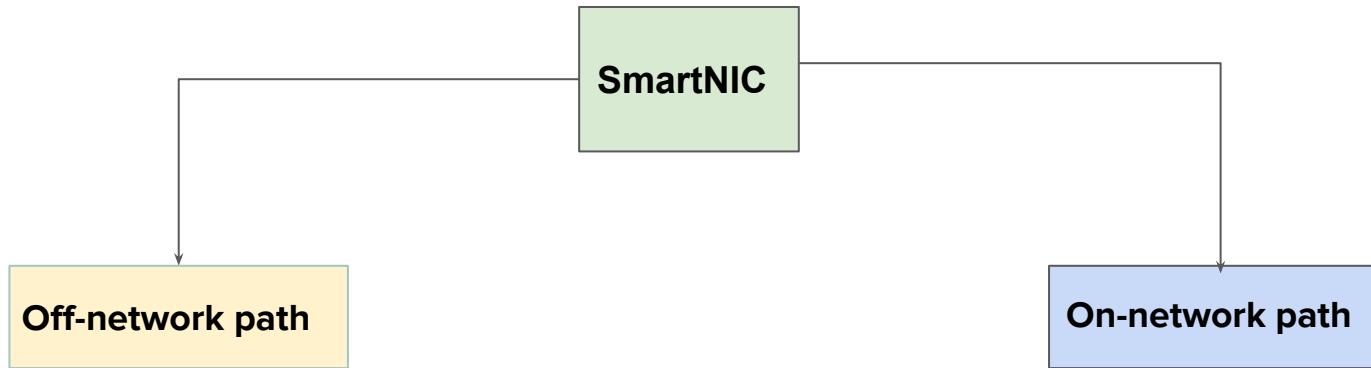
Offloading application to FPGA NIC



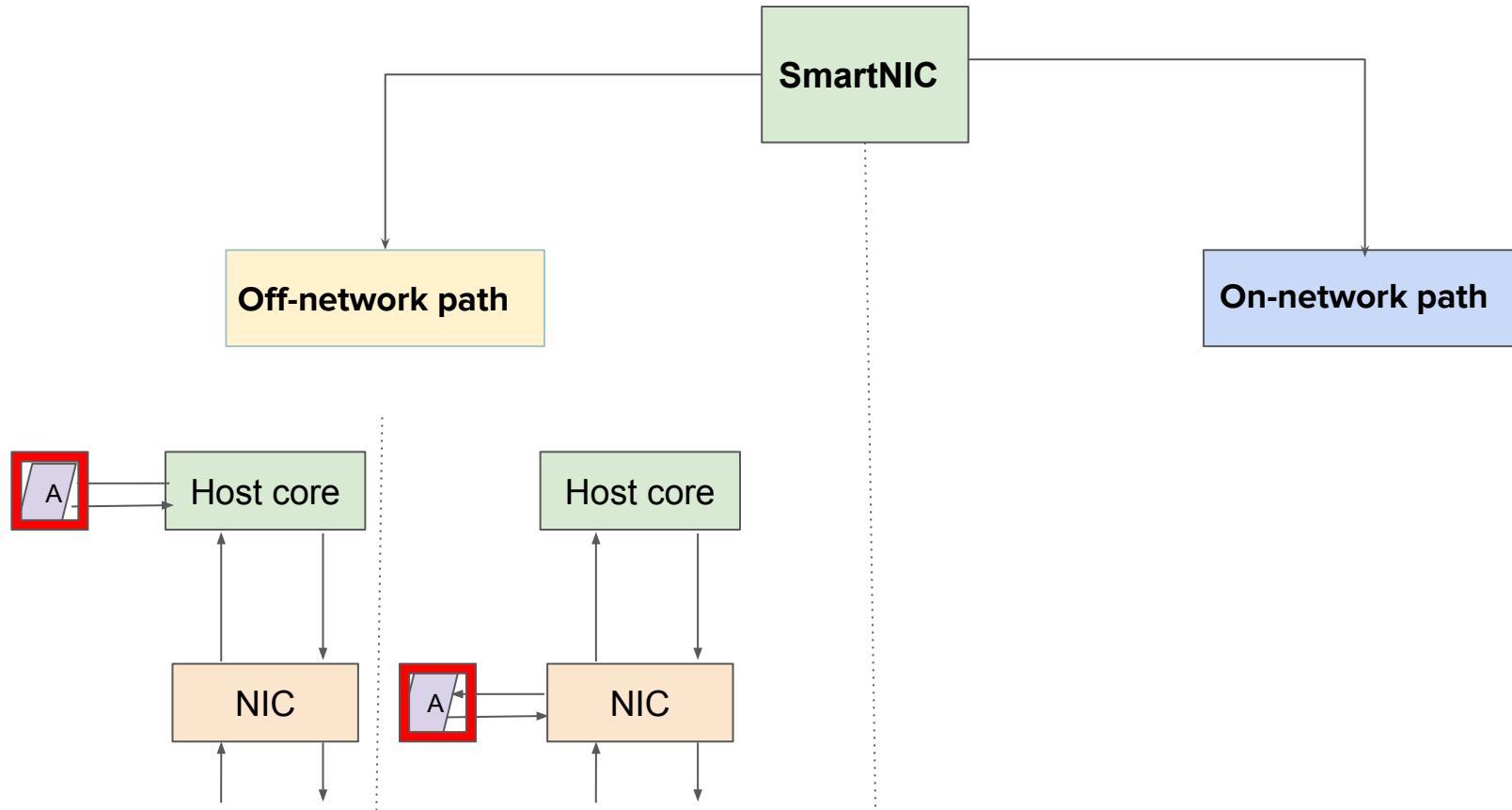
AES-GCM: Cryptography algorithm used in TLS.

Latency reduction
88.5% to 97.3%

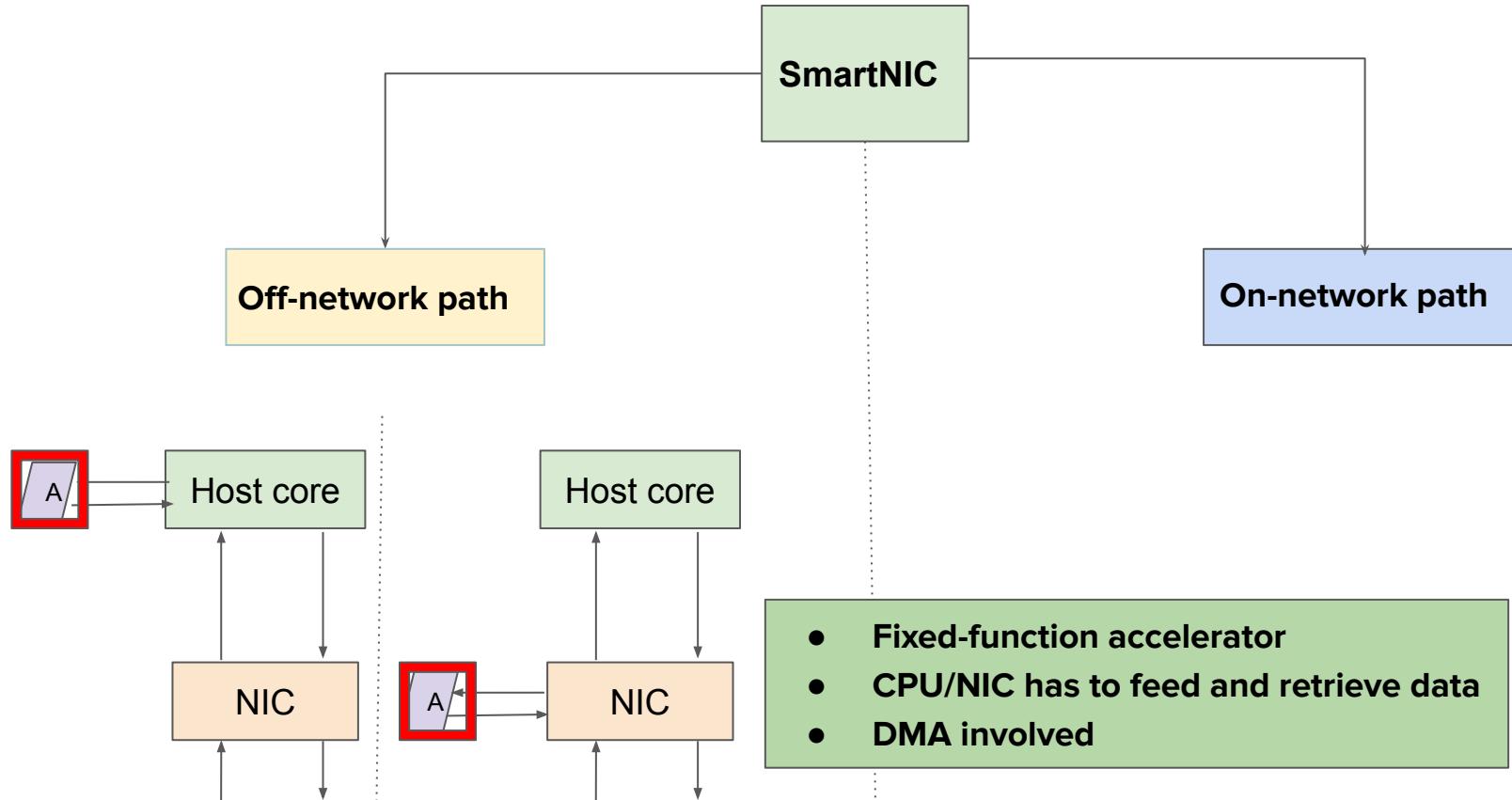
SmartNIC classification



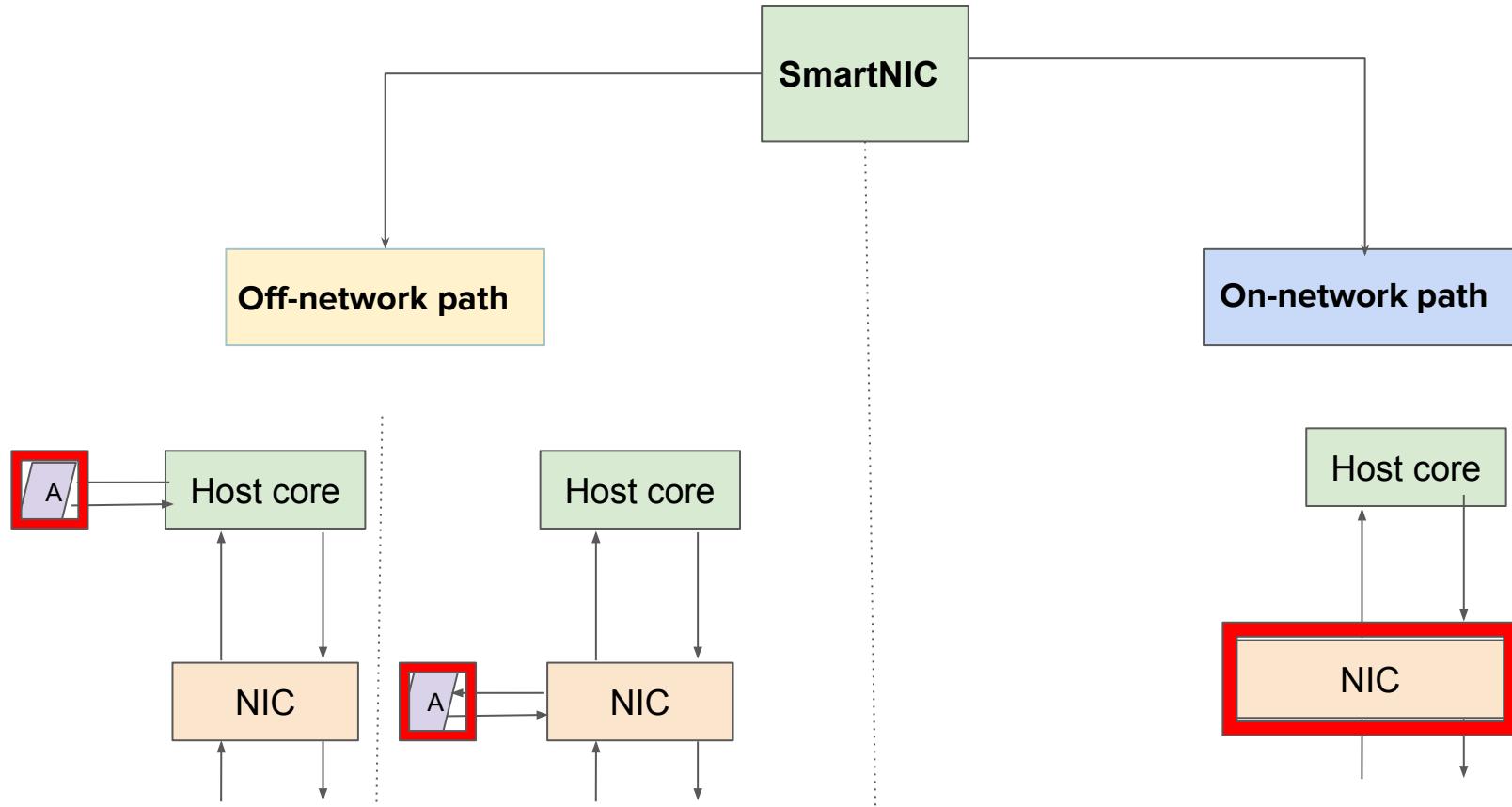
SmartNIC classification



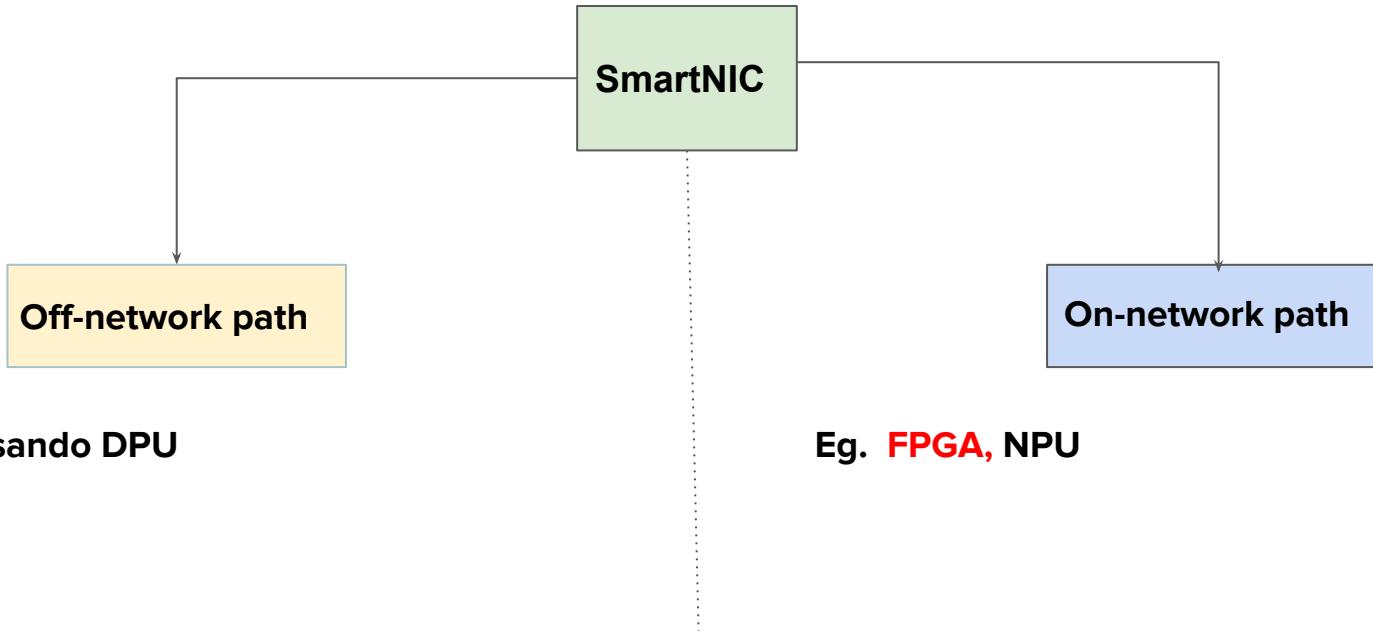
SmartNIC classification



SmartNIC classification



SmartNIC classification



Eg. Pensando DPU

Eg. **FPGA, NPU**

NPU-Network processing unit

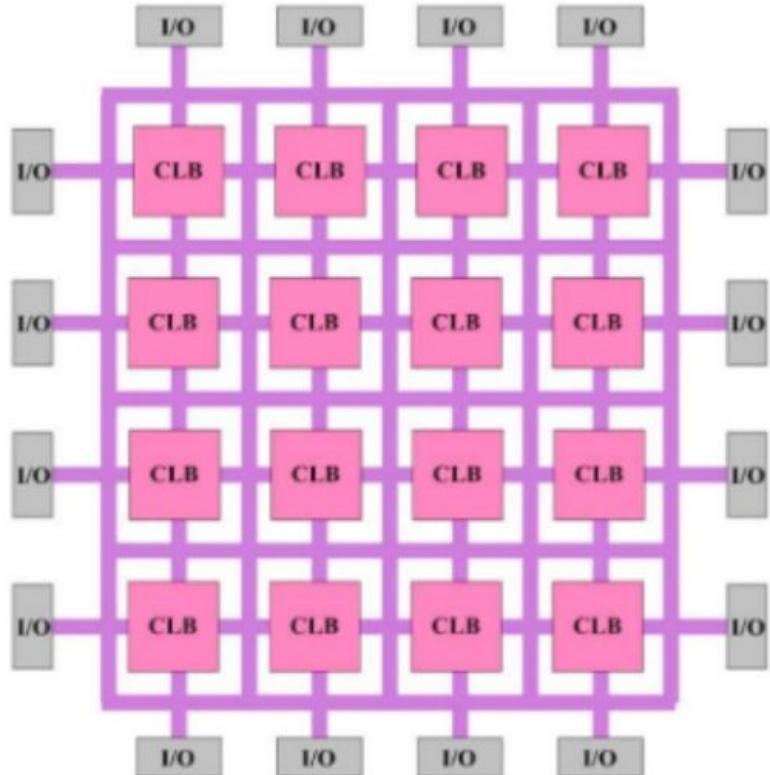
DPU-Data processing unit

FPGA- Field programmable gate array

Outline

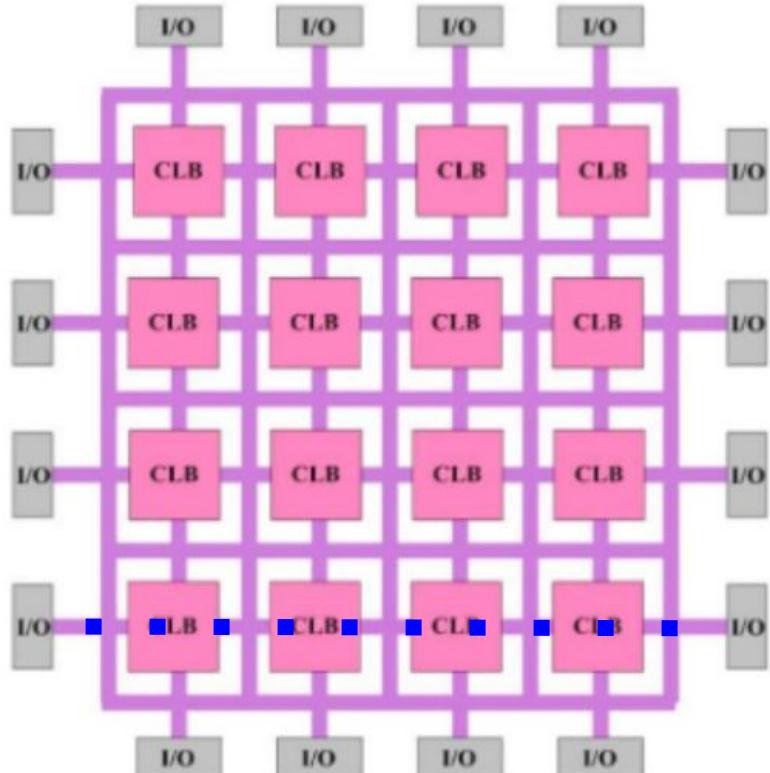
- Introduction: What is FPGA?
- FPGA theory (related to programming)
- FPGA programming tutorial
- Demonstration of “AES_GCM”
- Fun quiz

What is Field programmable gate array?



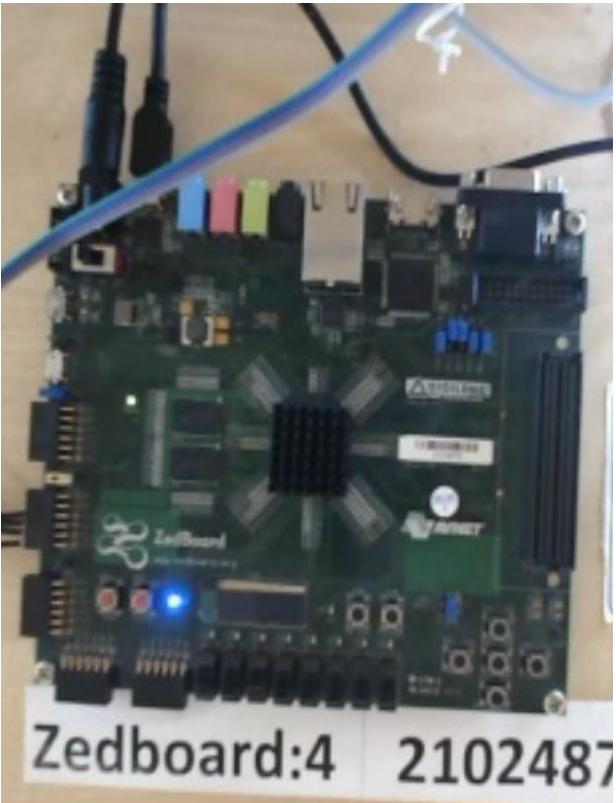
- It is a semiconductor device
- Consist of configurable logic block (CLB):
 - LUT, registers, multiplexer, adder, DSP units etc...
- CLB connected via programmable interconnect
- I/O is the input-output block

What is Field programmable gate array?



- It is a semiconductor device
- Consist of configurable logic block (CLB):
 - LUT, registers, multiplexer, adder, DSP units etc...
- CLB connected via programmable interconnect
- I/O is the input-output block

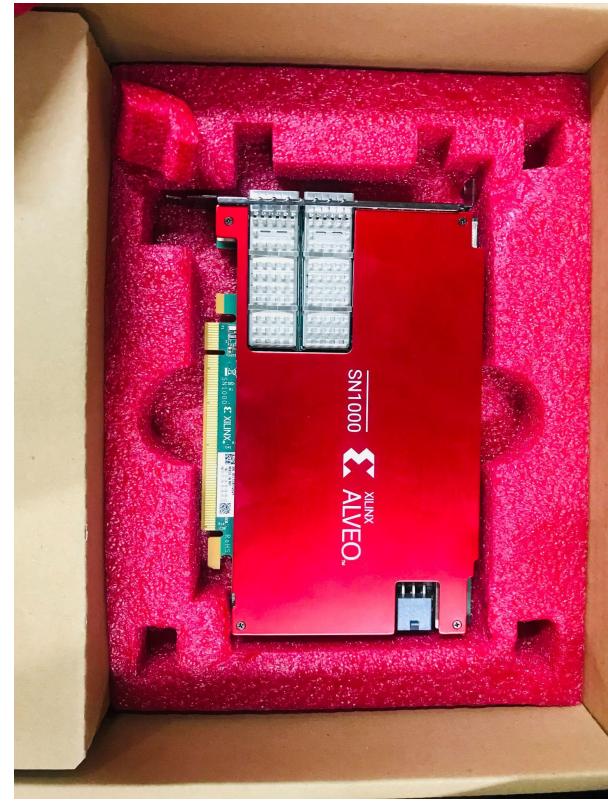
FPGA : Field programmable gate array



- Zedboard



- ZCU 106

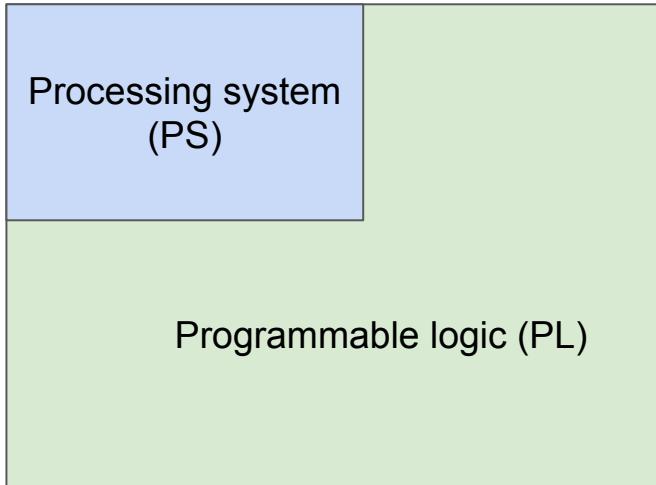


- Alevo SN1000

FPGA Applications

- 1) Data Centers and High-Performance Computing
- 2) Aerospace and Defense
- 3) Tele-communications processing
- 4) Industrial Automation... etc

FPGA: Field programmable gate array



- **Processing system (PS)**
 - ARM Dual cortex-A9 MPCore system
- **Programmable logic (PL)**
 - Configurable logic block (CLB)
 - Lookup tables (LUT)
 - Flip-flop (FF)
 - Block RAM (BRAM)

How to use FPGA

FPGA understand

- VHDL
- Verilog

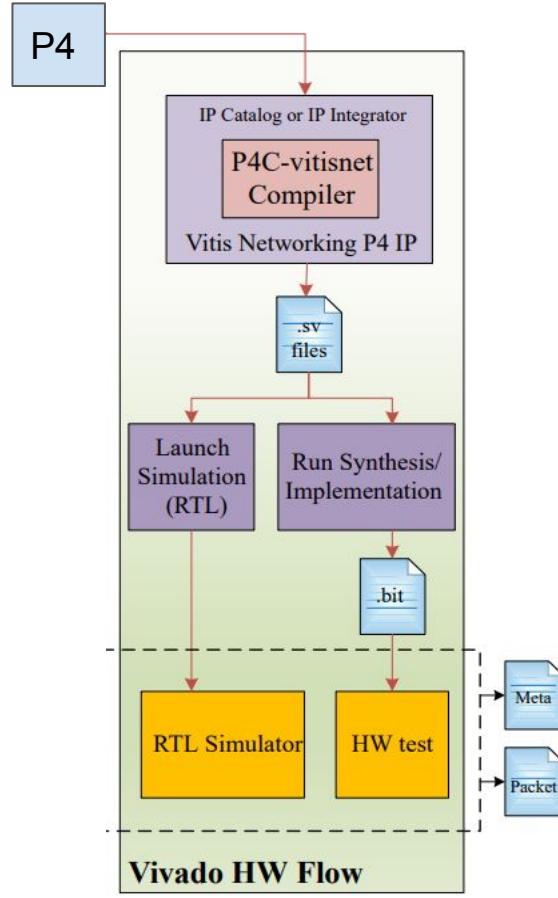
FPGA-NIC in practice

FPGAs can be integrated into SmartNICs to provide hardware acceleration for networking tasks

Eg.: Alevo SN1000



P4 to FPGA program



- Convert P4 code to FPGA code:
 - Vivado Hardware flow

P4 to FPGA program

Few **restrictions** in P4

- a) Division
- b) Modulus
- c) Floating point operation
- d) No support of complex operations

P4 to FPGA program

Few **restrictions** in P4

- a) Division
- b) Modulus
- c) Floating point operation
- d) No support of complex operations

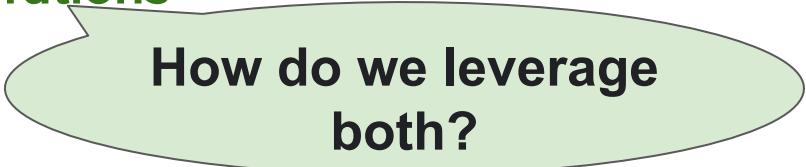
FPGA supports all such complex operations

P4 to FPGA program

Few **restrictions** in P4

- a) Division
- b) Modulus
- c) Floating point operation
- d) No support of complex operations

- **FPGA supports all such complex operations**
- **P4 provides ease of programming**



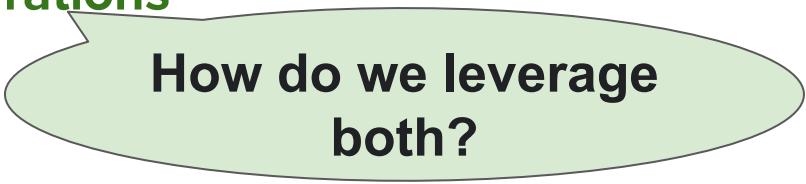
How do we leverage both?

P4 to FPGA program

Few **restrictions** in P4

- a) Division
- b) Modulus
- c) Floating point operation
- d) No support of complex operations

- **FPGA supports all such complex operations**
- **P4 provides ease of programming**



How do we leverage both?

Solution: By using **extern**

P4 to FPGA program

- Some **restriction** in p4
 - a) Multiplication
 - b) Modulus
 - c) Floating point operation
- FPGA supports->all such restrictions
- How to use?
- P4 ease of prg, and leverage complex op. Using FPGA
- How do we leverage flexibility and complex processing?
- -> extern

P4 to FPGA program

Few **restrictions** in P4

- a) Multiplication
- b) Modulus
- c) Floating point operation
- d) No support of complex operations

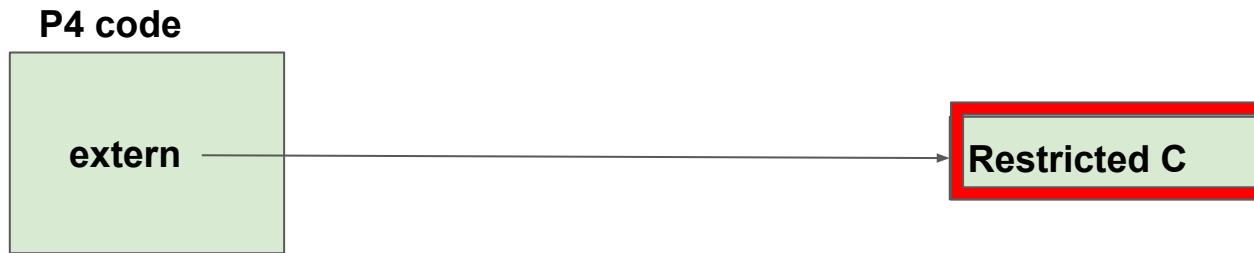


Eg. Cryptography algorithm, video compression, ...

P4 to FPGA program

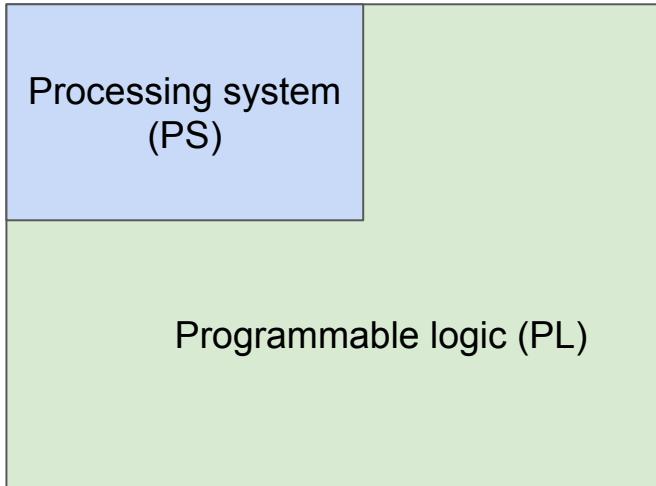
Few **restrictions** in P4

- a) Multiplication
- b) Modulus
- c) Floating point operation
- d) No support of complex operations



Eg. Cryptography algorithm, video compression...

FPGA: Field programmable gate array



- **Processing system (PS)**
 - ARM Dual cortex-A9 MPCore system
- **Programmable logic (PL)**
 - Configurable logic block (CLB)
 - Lookup tables (LUT)
 - Flip-flop (FF)
 - Block RAM (BRAM)

How to use FPGA

FPGA understand

- VHDL
- Verilog

How to use FPGA

FPGA understand

- VHDL
- Verilog

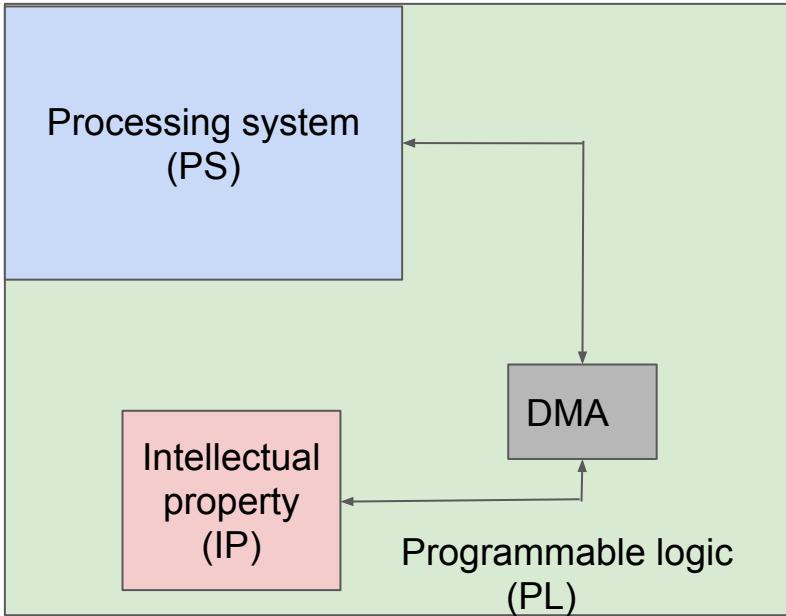
Developer has created a tool chain that will convert C language code to VHDL/verilog:



FPGA theory

- Basic understanding of FPGA components and their usage:
 - PS, PL
 - DMA
 - Memories: DRAM, BRAM, URAM
 - Designing of ports/interfaces
 - Stream interface, Memory mapped interface

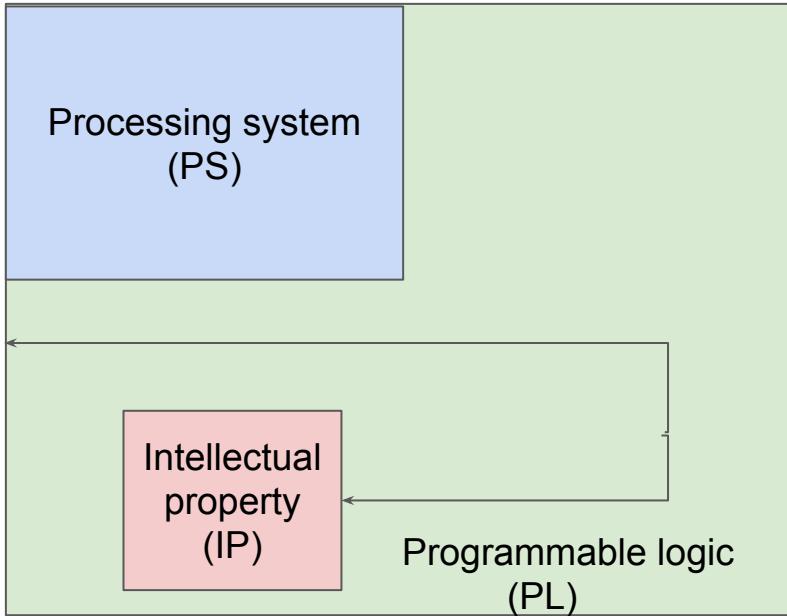
FPGA end-to-end working



- **IP is burnt in PL**
- **PS feeds the data to the IP via DMA, retrieve the result back from the IP via the DMA.**

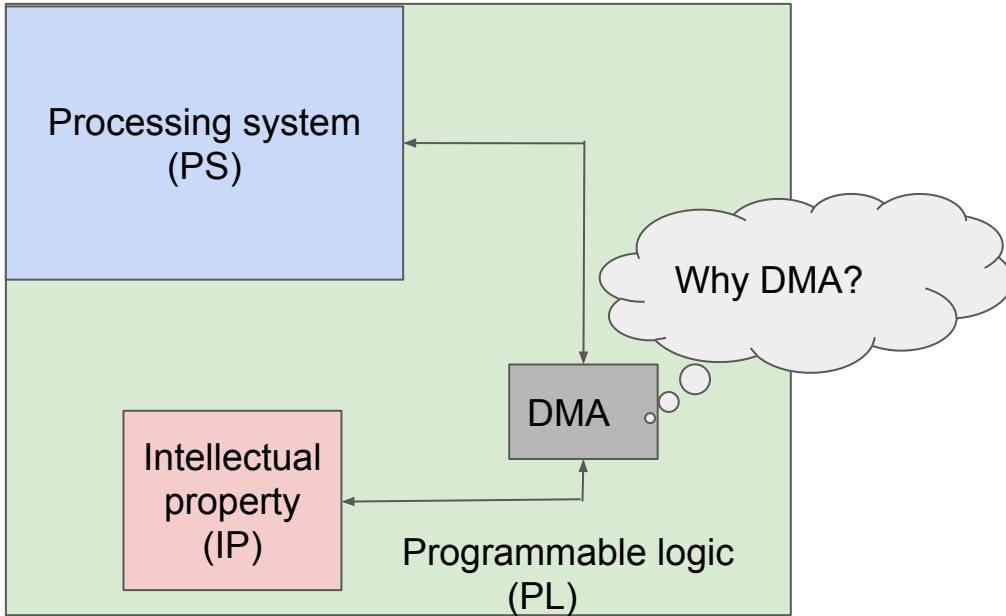
*DMA- Direct memory access

FPGA end-to-end working



- **IP is burnt in PL**
- **Data is coming from PL itself**

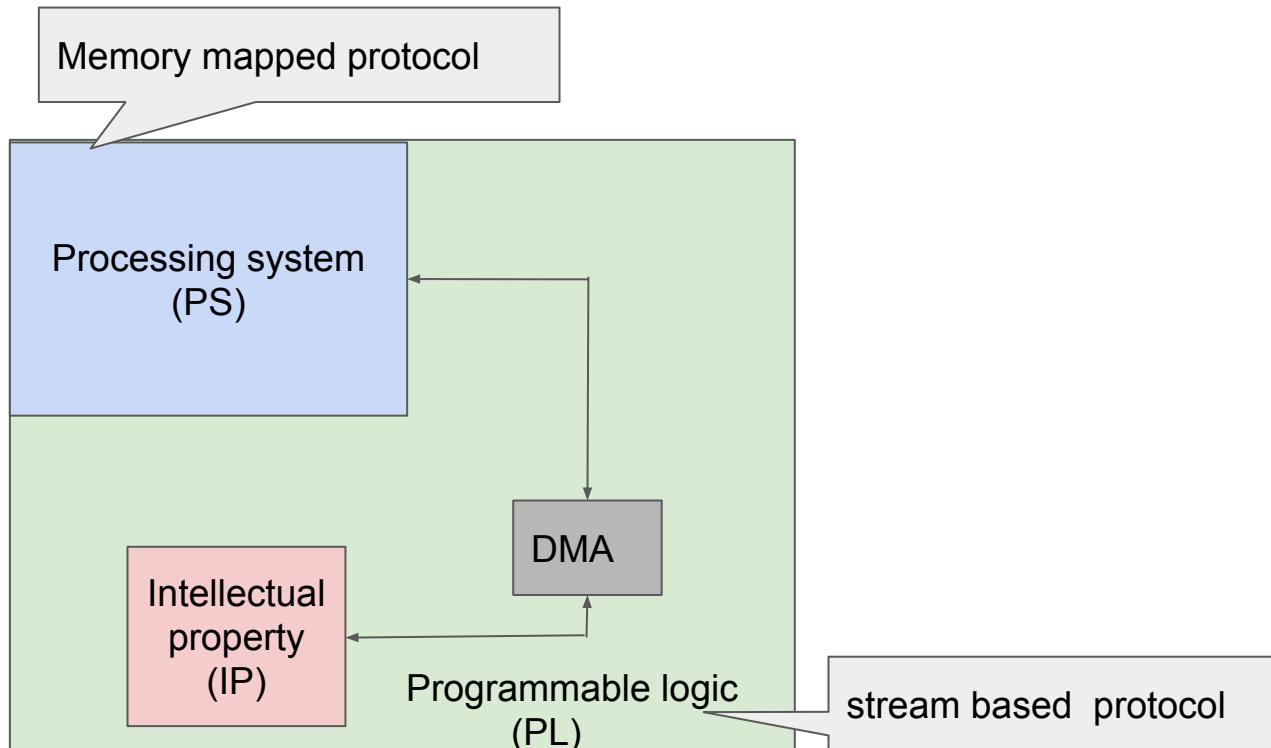
FPGA end-to-end working



- **IP is burnt in PL**
- **PS feeds the data to the IP via DMA, retrieve the result back from the IP via the DMA.**

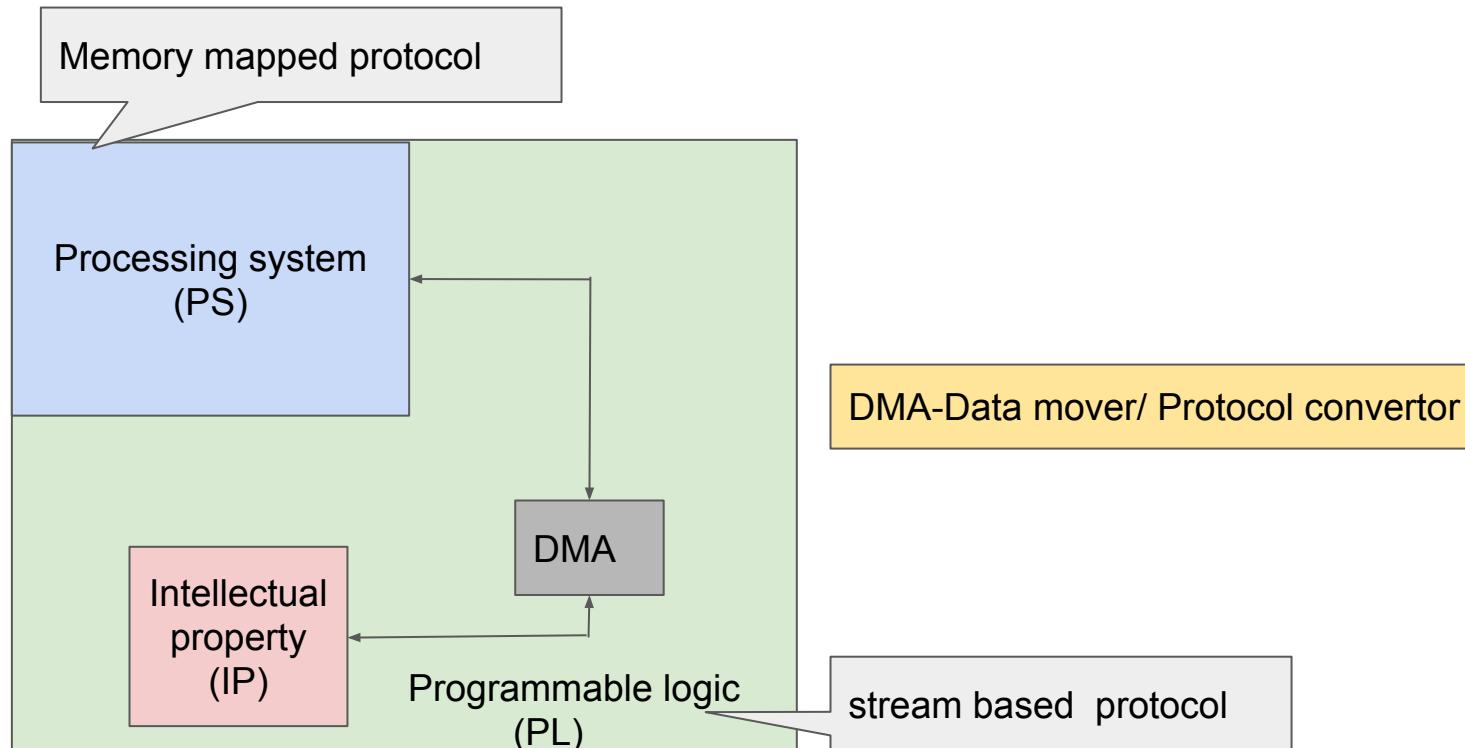
*DMA- Direct memory access

FPGA end-to-end working



*DMA- Direct memory access

FPGA end-to-end working

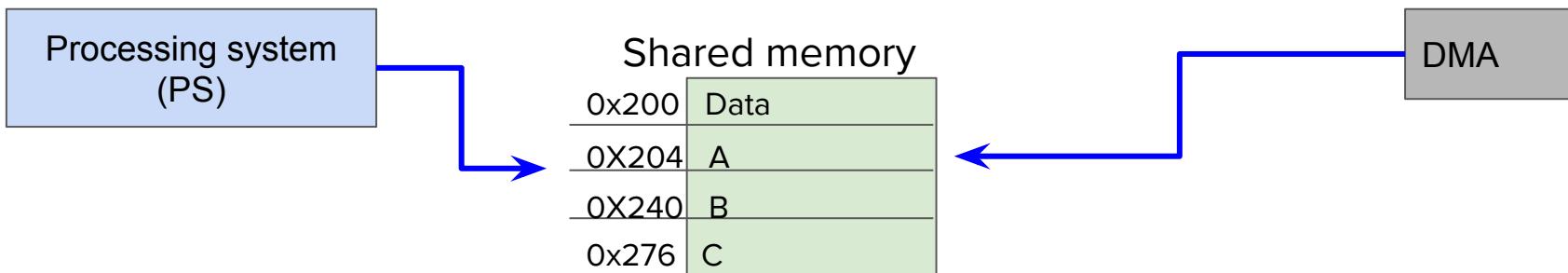


*DMA- Direct memory access

Memory mapped (mm) protocol

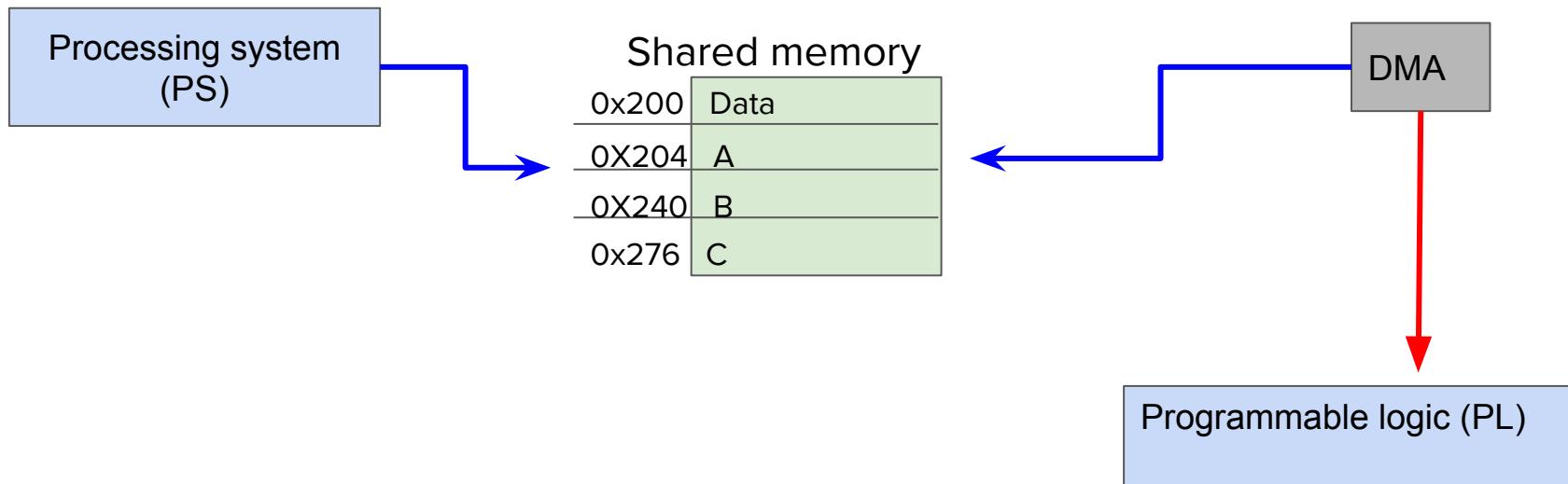
- Shared memory access between processing system and DMA
- Data and its corresponding addresses.

→ mm



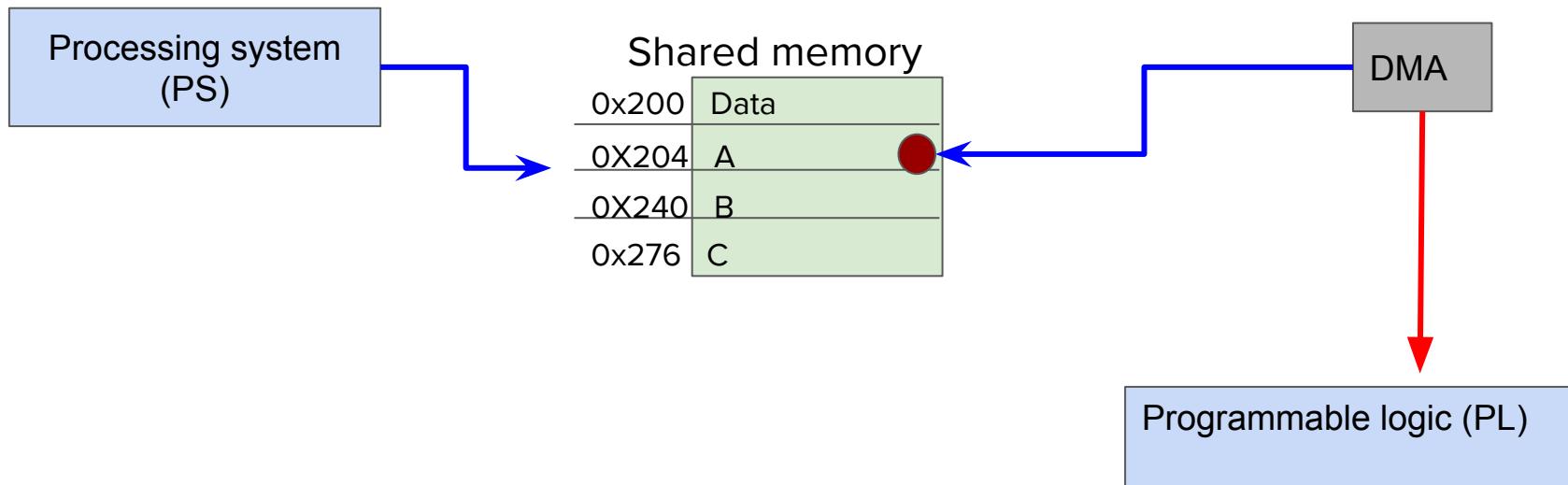
Stream based protocol

- Transfer data as continuous stream, contain data only, no addresses.



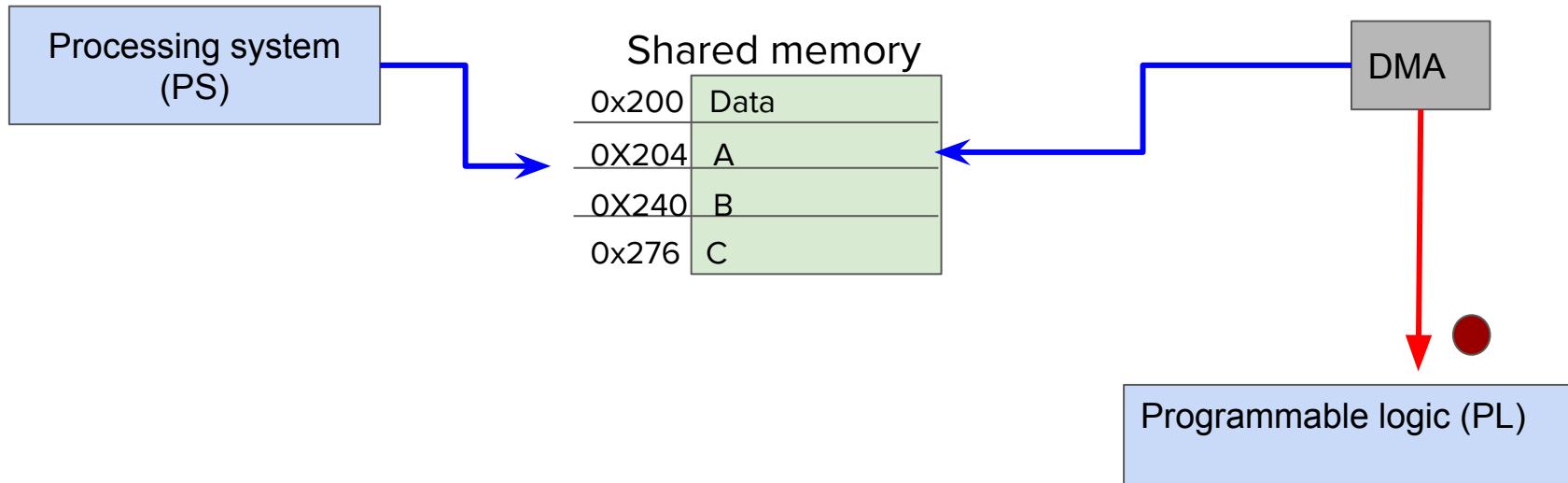
Stream based protocol

- Transfer data as continuous stream, contain data only, no addresses.



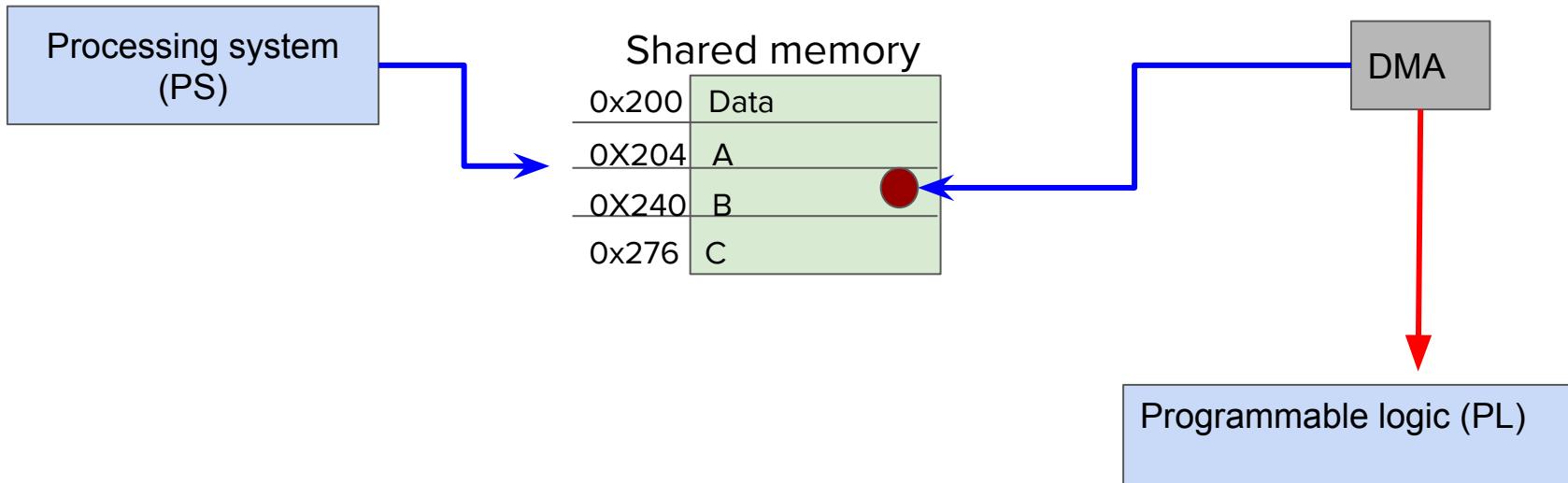
Stream based protocol

- Transfer data as continuous stream, contain data only, no addresses.



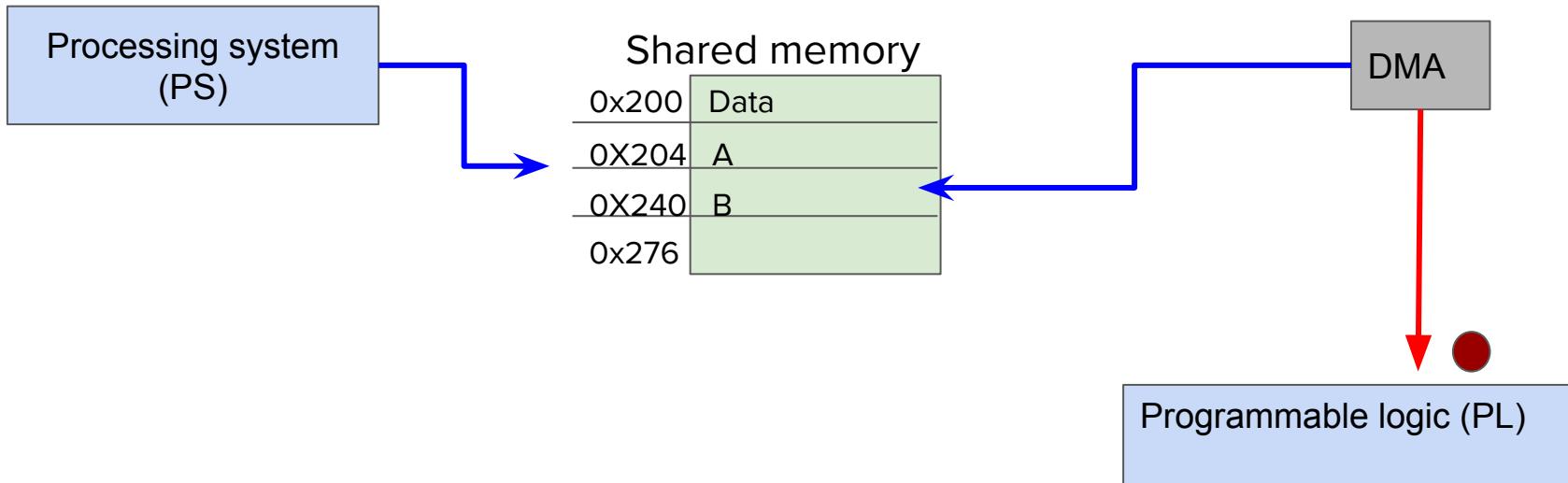
Stream based protocol

- Transfer data as continuous stream, contain data only, no addresses.



Stream based protocol

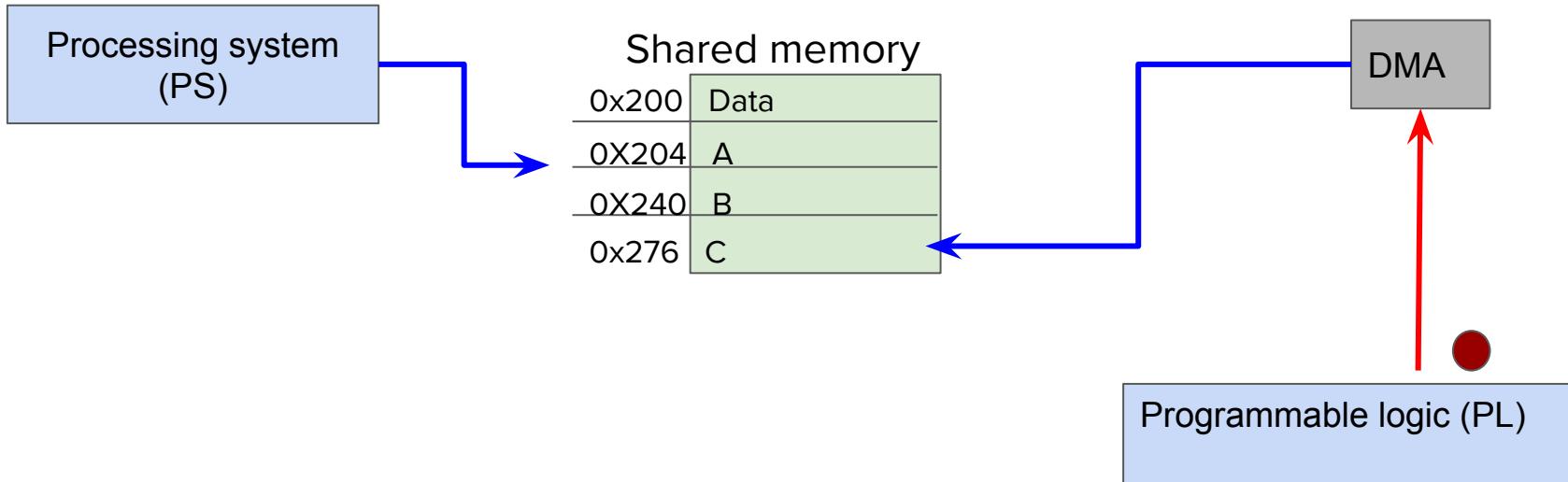
- Transfer data as continuous stream, contain data only, no addresses.



Stream based protocol

- Transfer data as continuous stream, contain data only, no addresses.

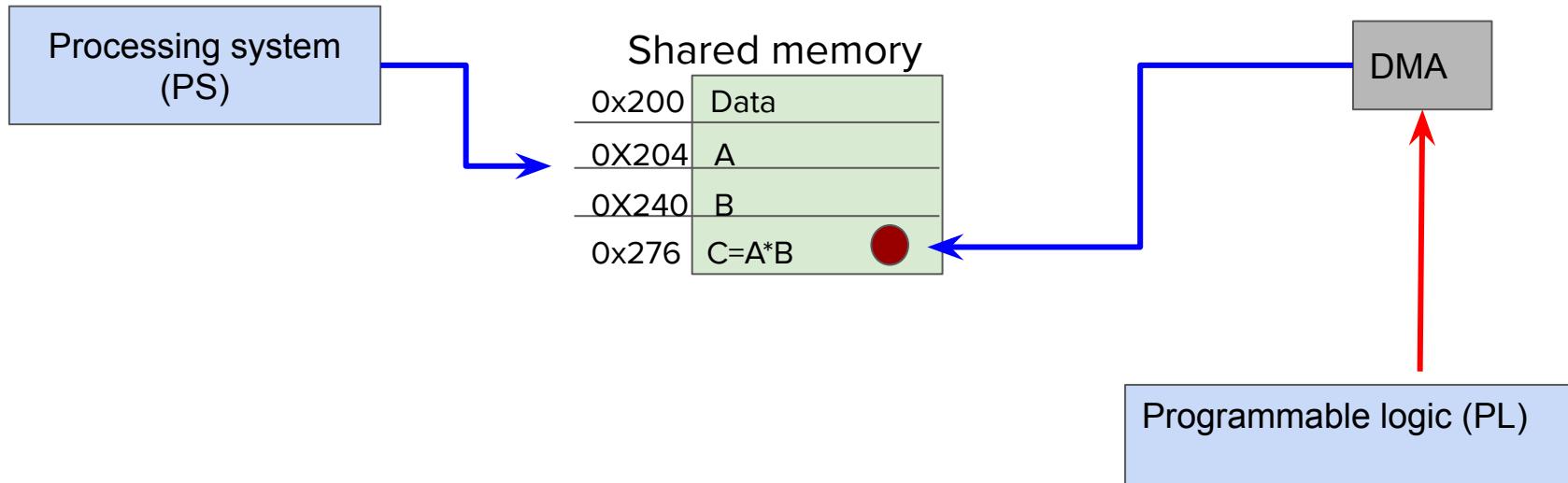
● Data
mm → Stream



Stream based protocol

- Transfer data as continuous stream, contain data only, no addresses.

● Data
mm → Stream



What is memory mapped and stream based protocol?

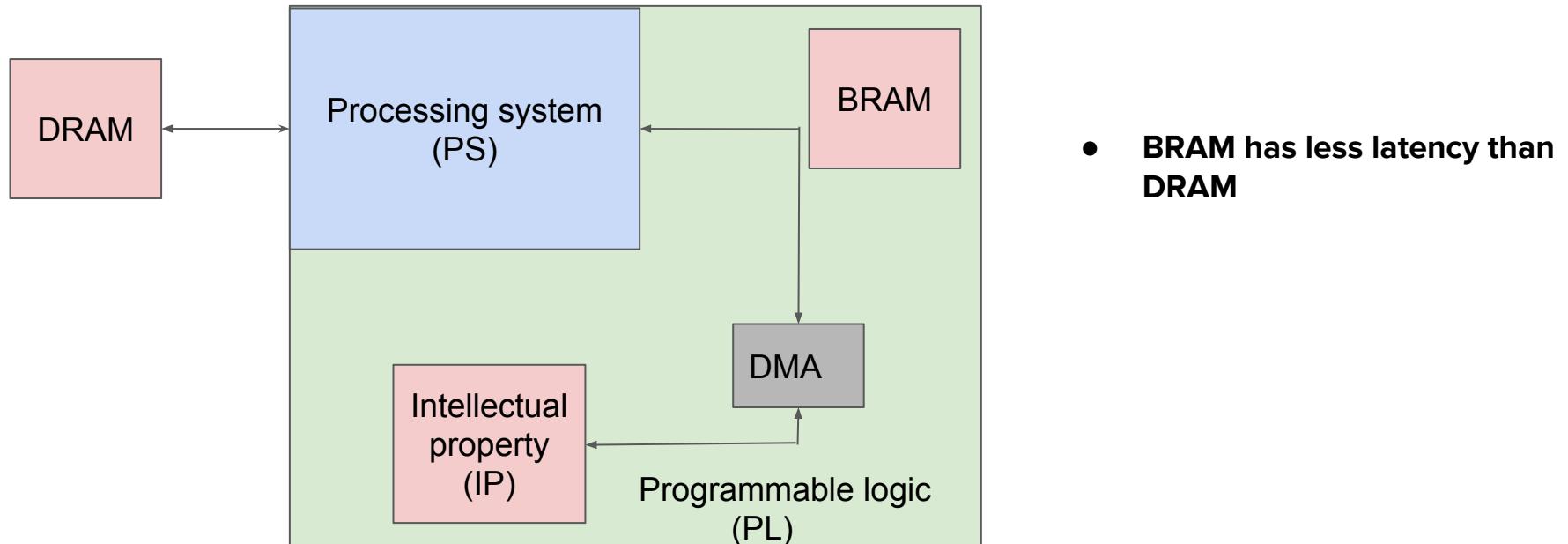
Memory mapped

- Data and its corresponding addresses.
- Shared memory access

Stream based

- Transfer data as continuous stream
- Contains data only.

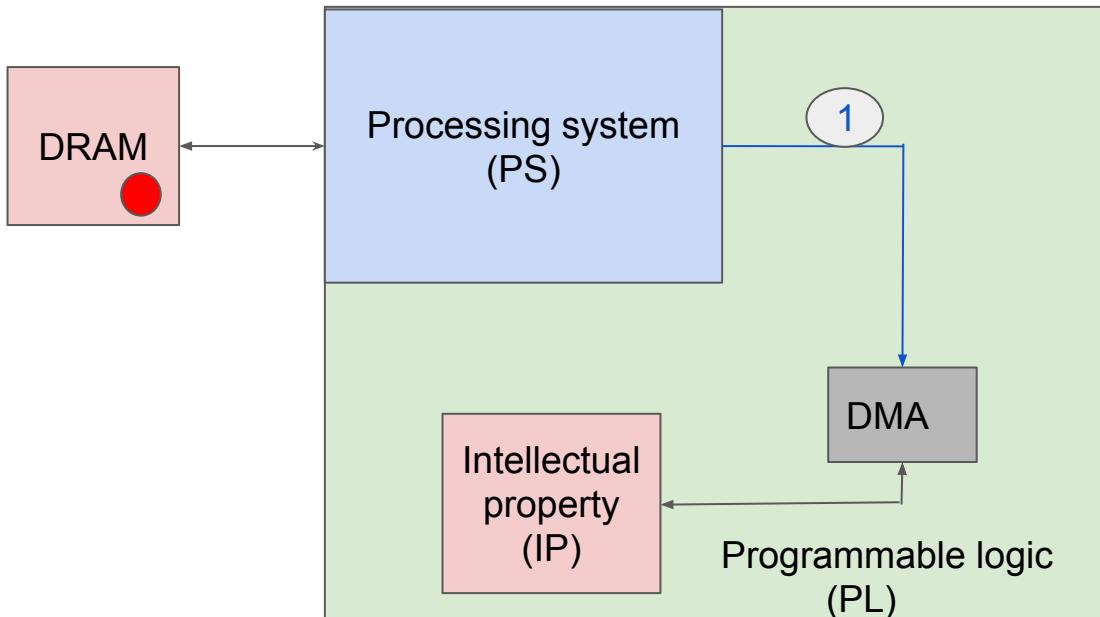
FPGA: Memories



*DMA- Direct memory access

FPGA end-to-end flow of data

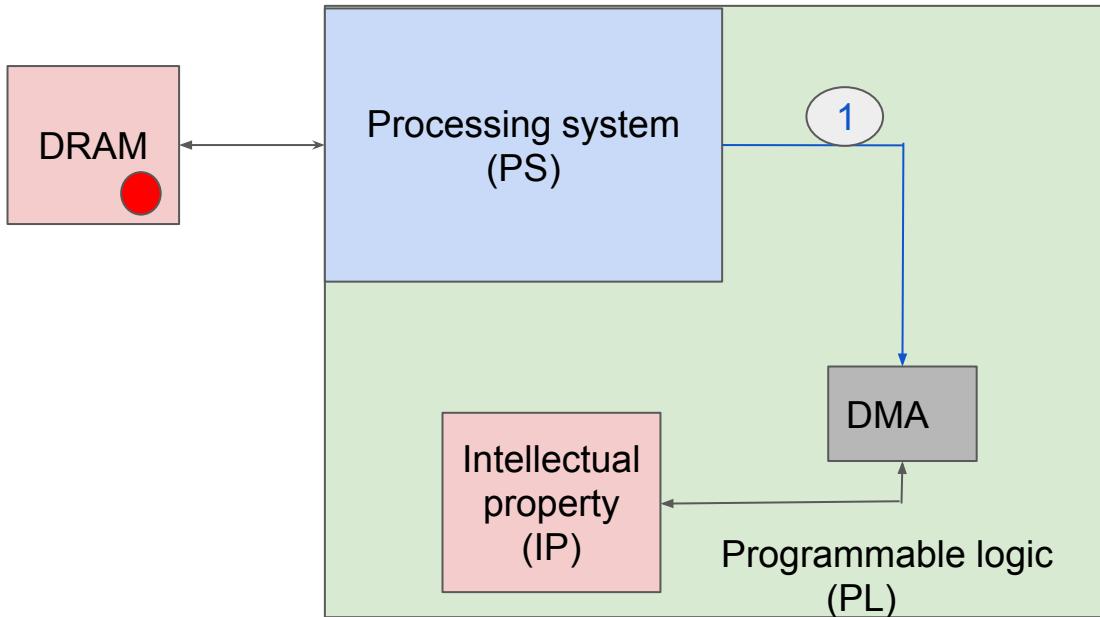
● Data



*DMA- Direct memory access

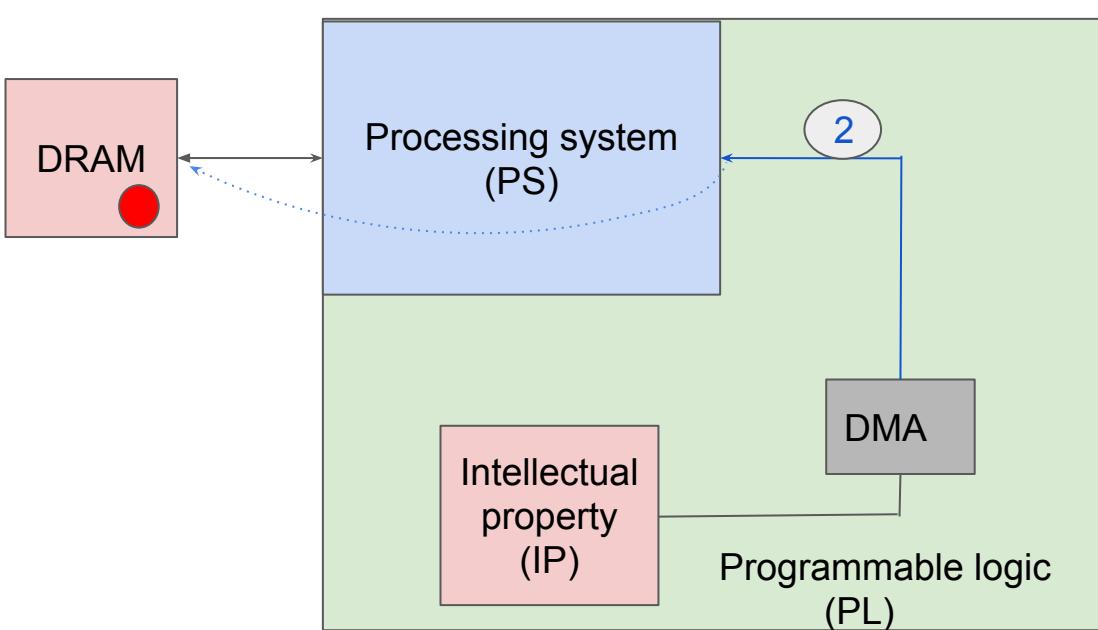
FPGA end-to-end flow of data

● Data



- 1) PS -> Configure DMA AXI_LITE

FPGA end-to-end flow of data

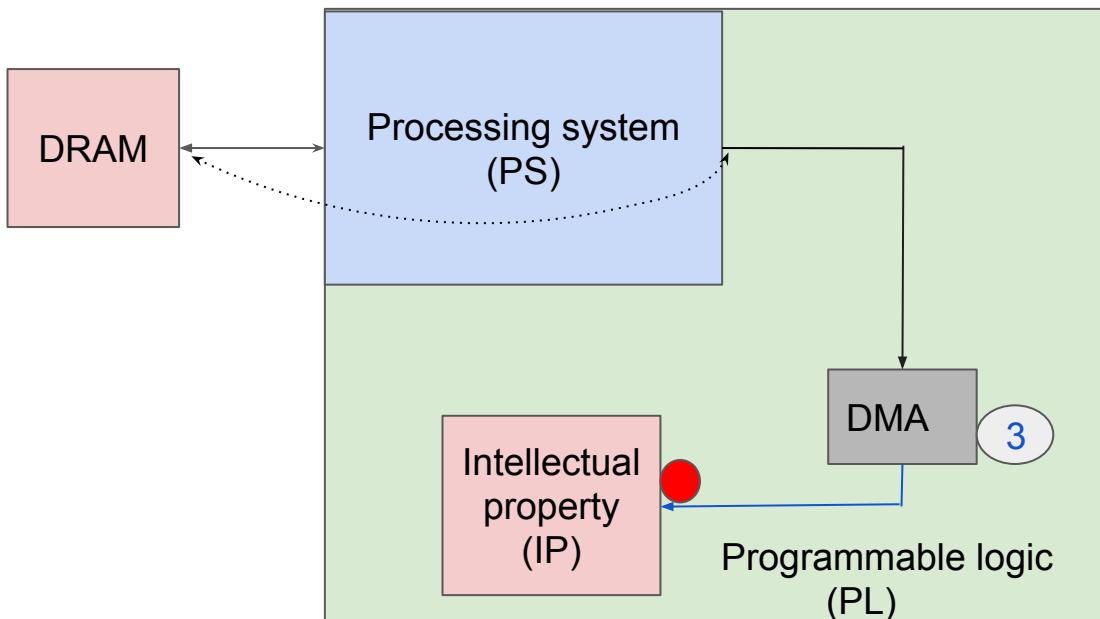


2) PS (&data) -> DMA

DMA will fetch the data from memory

FPGA end-to-end flow of data

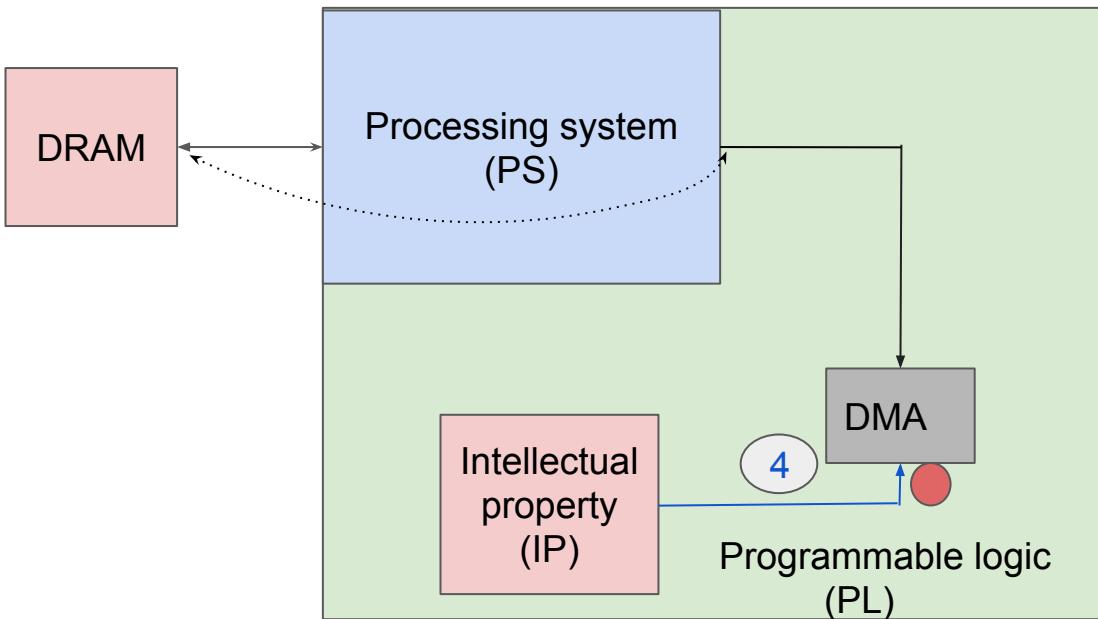
● Data



3) DMA(data) -> IP

DMA: Supply data to the IP

FPGA end-to-end flow of data

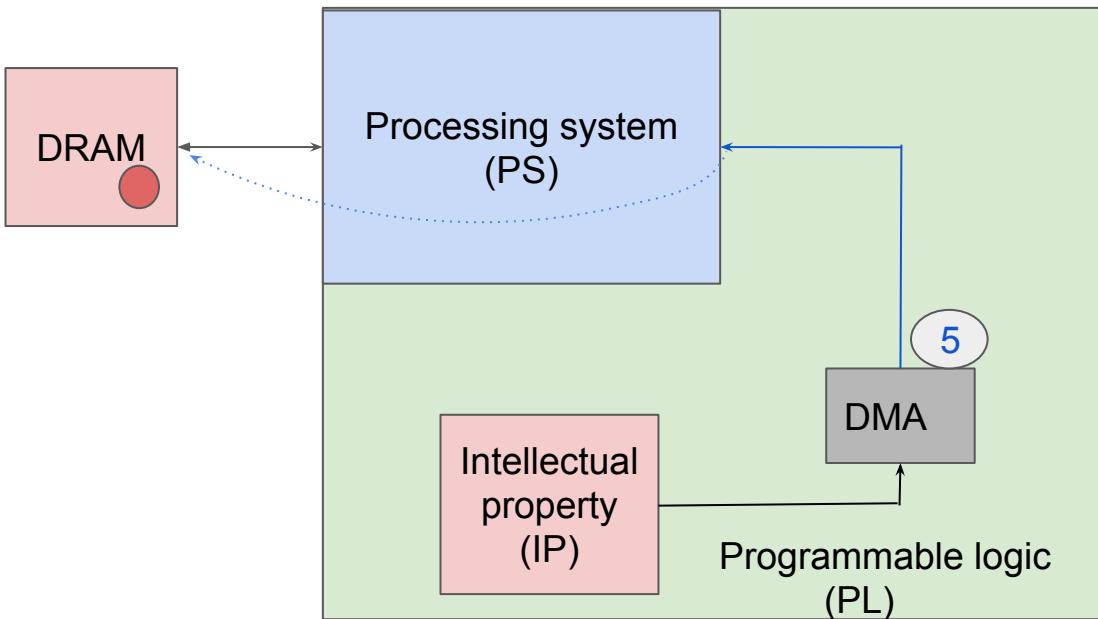


Data

4) IP (data*) -> DMA

IP sent the data back to the DMA, after computation

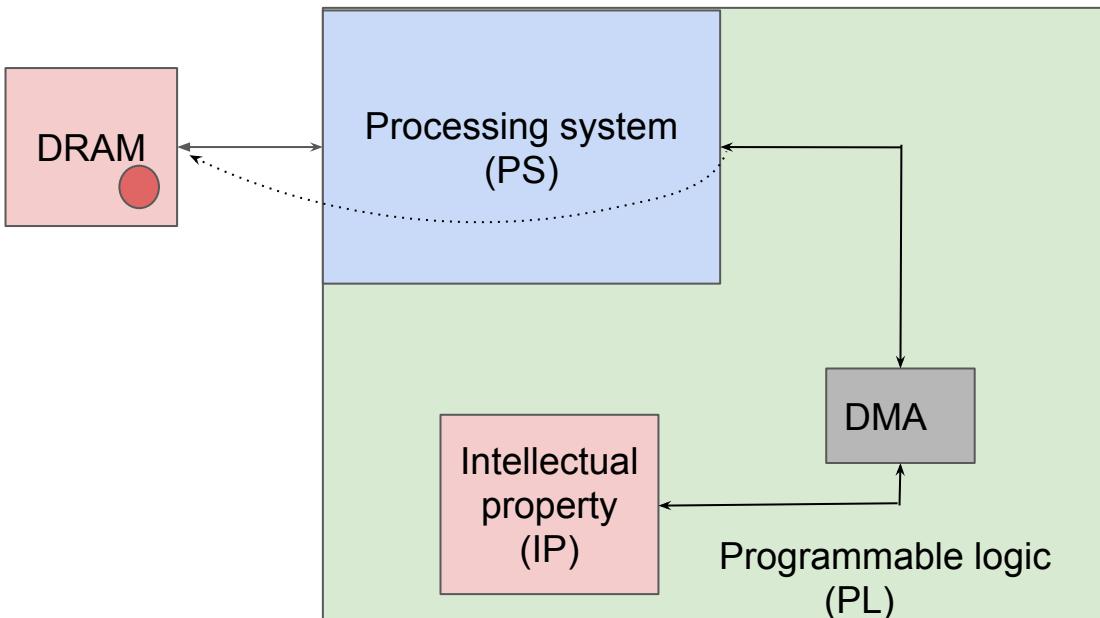
FPGA end-to-end flow of data



5) DMA (data)-> DRAM
And inform the (&data) to PS.



FPGA end-to-end flow of data



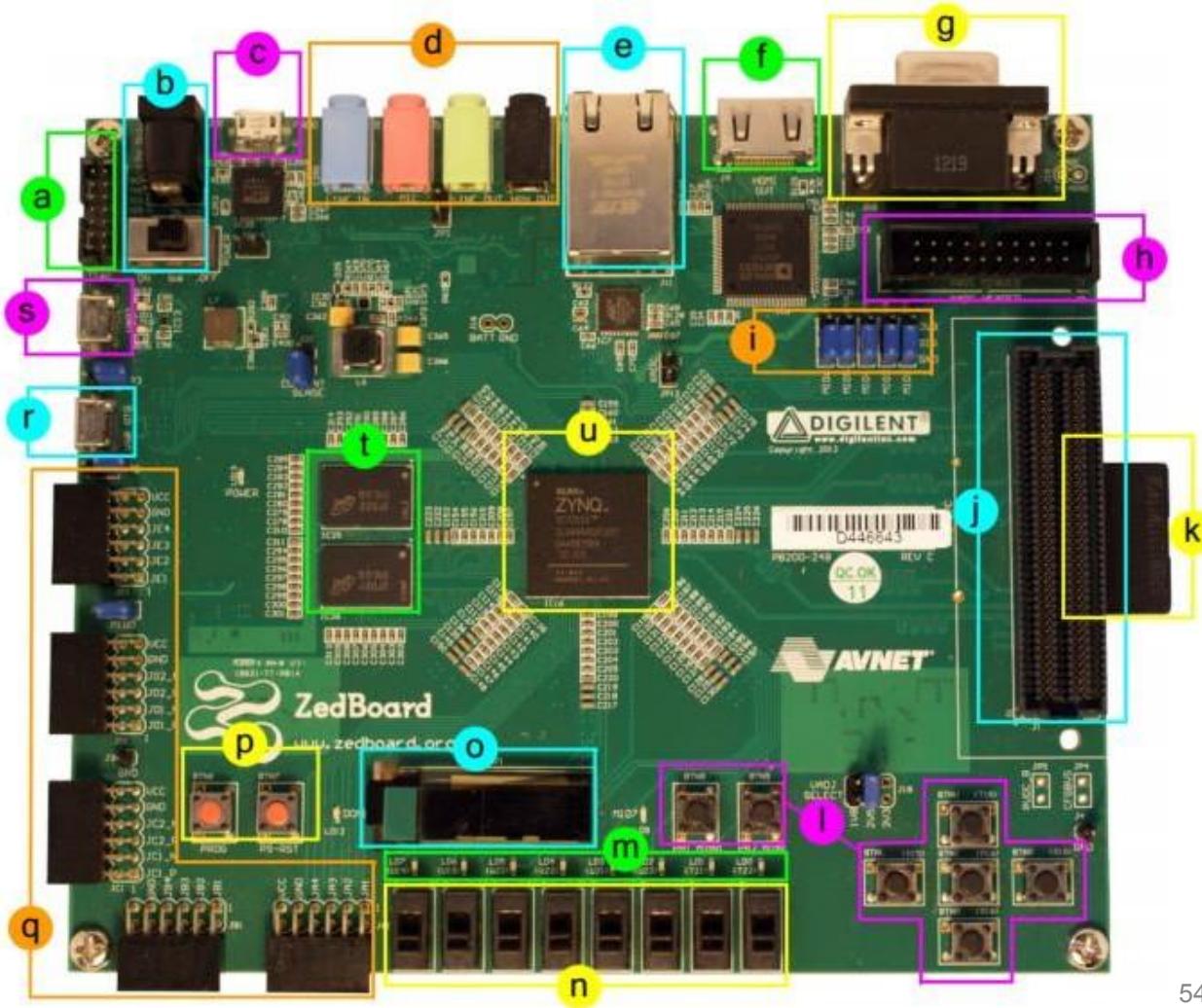
● Data

- 1) PS -> Configure DMA
- 2) PS (&data) -> DMA
- 3) DMA(data) -> IP
- 4) IP (data*) -> DMA
- 5) DMA (data)-> DRAM
and inform the (&data)
to PS.

*DMA- Direct memory access

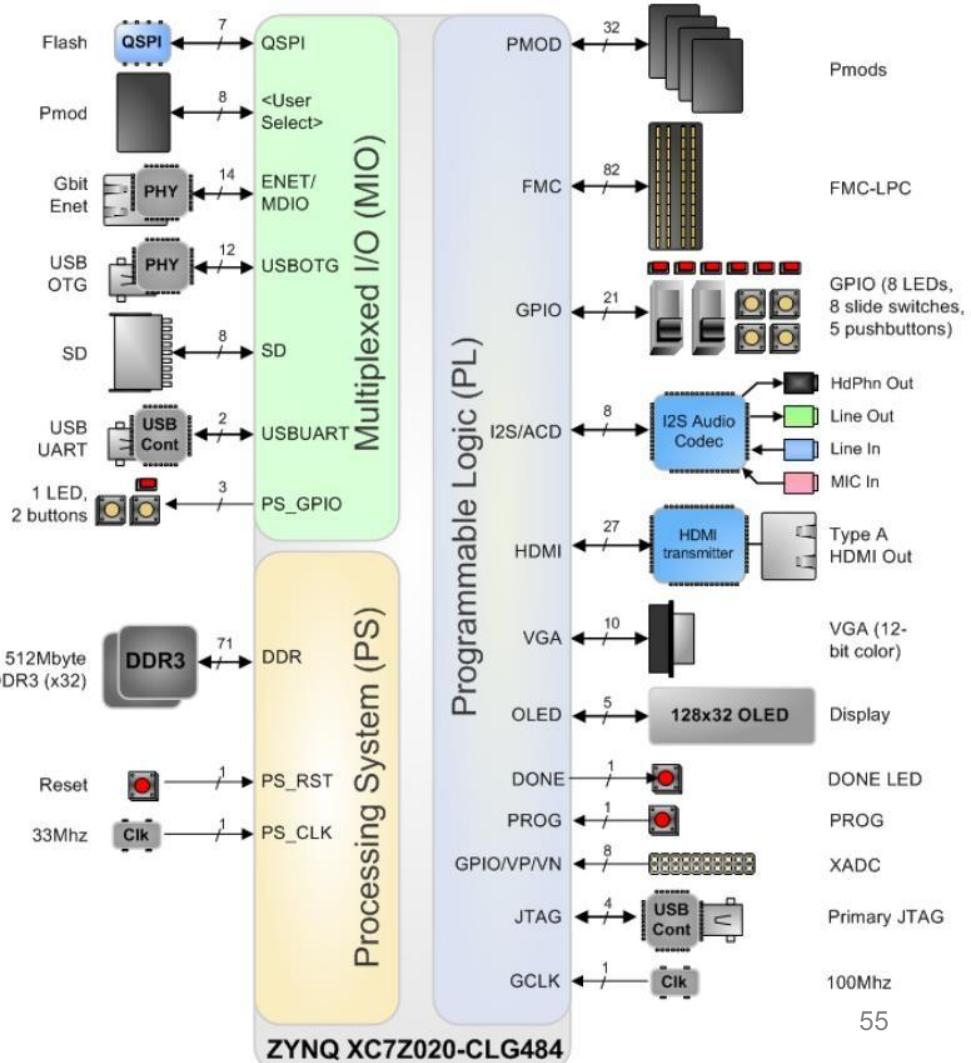
Zedboard

- a Xilinx JTAG connector
- b Power input and switch
- c USB-JTAG (programming)
- d Audio ports
- e Ethernet port
- f HDMI port (output)
- g VGA port
- h XADC header port
- i Configuration jumpers
- j FMC connector
- k SD card (underside)
- l User push buttons
- m LEDs
- n Switches
- o OLED display
- p Prog & reset push buttons
- q 5 x Pmod connector ports
- r USB-OTG peripheral port
- s USB-UART port
- t DDR3 memory
- u Zynq device (+ heatsink)

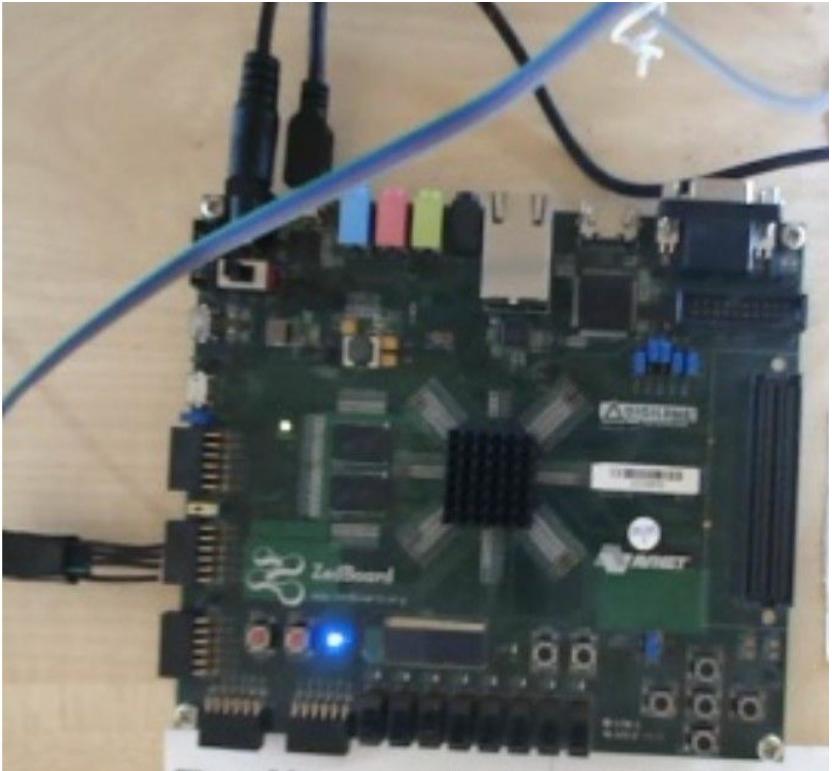


Zedboard

- Processor
 - Zynq™-7000 AP SoC XC7Z020-CLG484-1
- Memory
 - 512 MB DDR3
 - 256 Mb Quad-SPI Flash
 - 4 GB SD card
- Communication
 - Onboard USB-JTAG Programming
 - o 10/100/1000 Ethernet
 - USB OTG 2.0 and USB-UART
- Expansion connectors
 - FMC-LPC connector (68 single-ended or 34 differential I/Os)
 - 5 Pmod™ compatible headers (2x6)
 - Agile Mixed Signaling (AMS) header
- Clocking
 - 33.33333 MHz clock source for PS
 - 100 MHz oscillator for PL
- Display
 - HDMI output supporting 1080p60 with 16-bit, YCbCr, 4:2:2 mode color
 - VGA output (12-bit resolution color)
 - 128x32 OLED display
- Configuration and Debug
 - Onboard USB-JTAG interface
 - Xilinx Platform Cable JTAG connector
- General Purpose I/O
 - 8 user LEDs
 - 7 push buttons
 - 8 DIP switches



FPGA board



Zedboard



ZCU 106

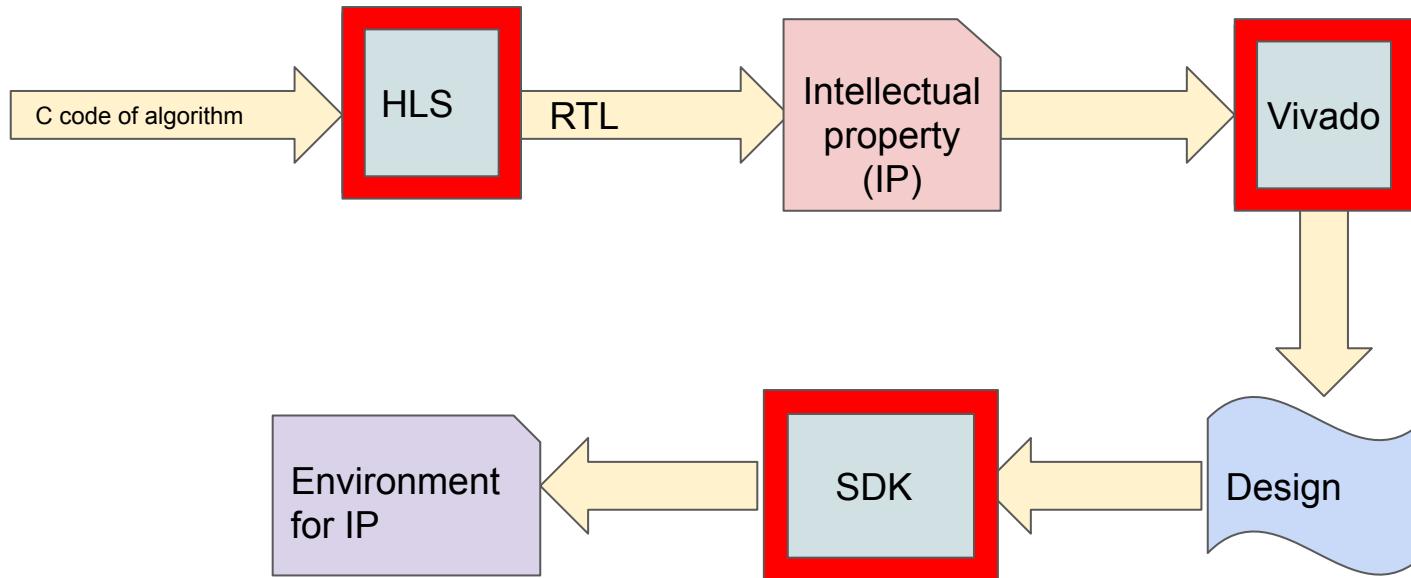
FPGA programming tutorial

- Github repository:
 \$ git clone <https://github.com/pnl-iiitd/2023-P4-for-all-tutorial.git>
- \$ source /tools/Xilinx/Vivado/2019.1/settings64.sh
- \$ vivado &
- \$ vivado_hls

FPGA programming tutorial

- Systematic process of writing program on FPGA:
 - a) HLS
 - b) Vivado
 - c) SDK

FPGA: Workflow



Cont'd

- Program of :
 - a) Finding square of no.
 - b) Matrix multiplication
- How to make code parallelizable:
->By introduction of pragma

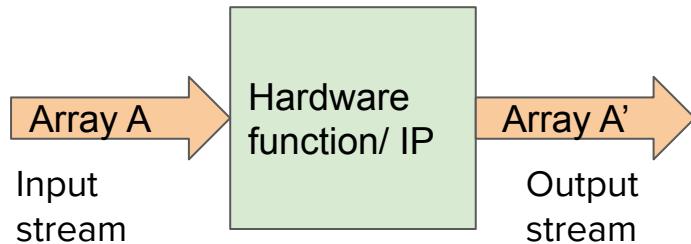
Restriction in HLS

- 1) Function return type will always be void
- 2) No use of while loop(un-deterministic) loop.
- 3) Recursion is not allowed.
- 4) No concept of dynamic memory allocation.
- 5) No other libraries are allowed other than #include<math.h>, #include<stdio.h>

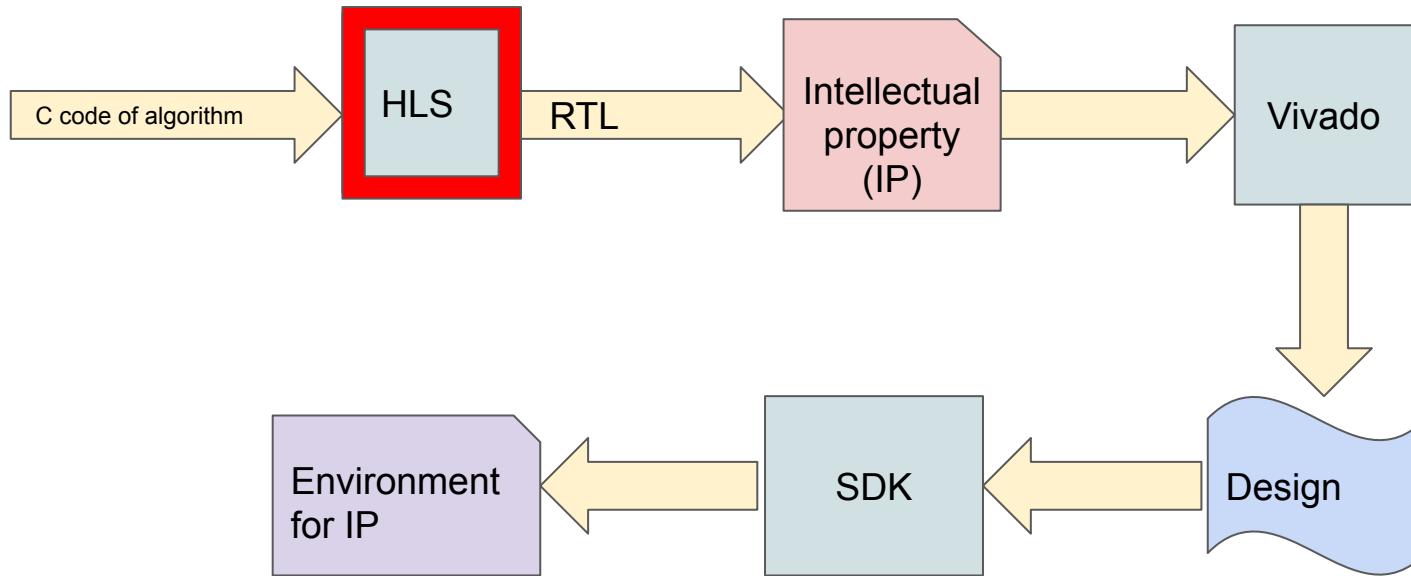
FPGA hands-on

Square of number

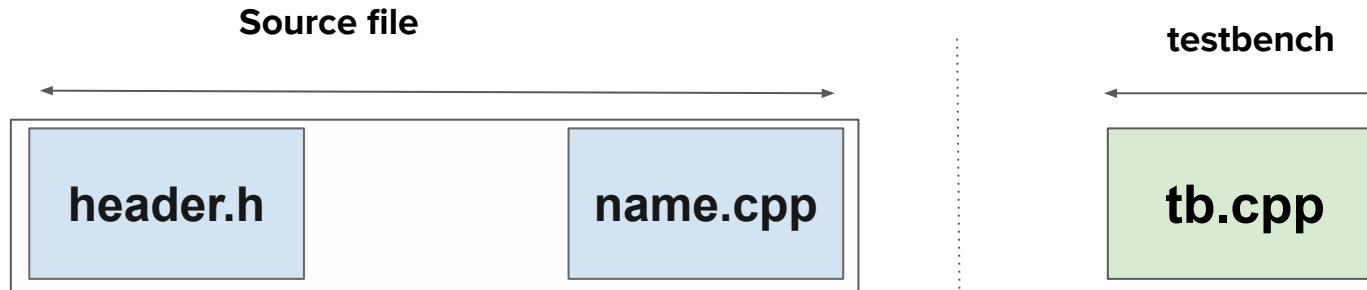
- Input: An array of numbers say 1 to 10 i.e 1, 2, 3, 4, 5, 6, 7, 8, 9.
- Output: An array of square of given numbers i.e 1, 4, 9, 16, 25, 36, 49, 64, 81.



FPGA: Workflow



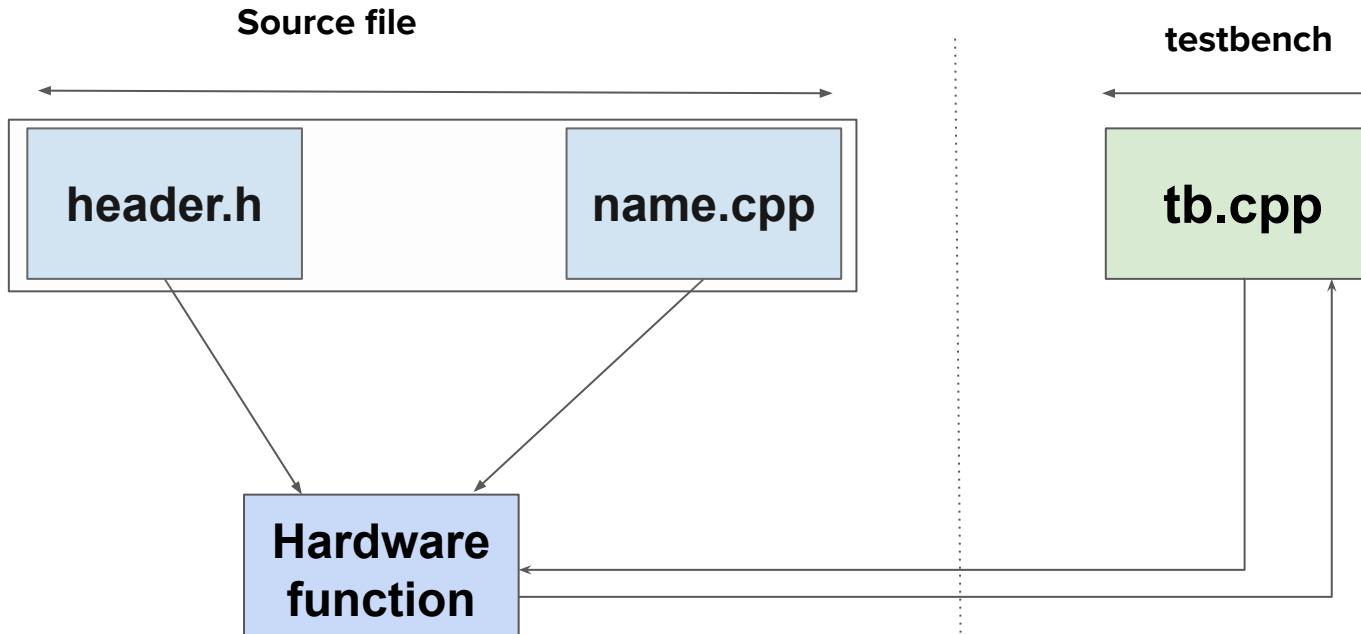
HLS: Higher level synthesis



- 1) Main function
- 2) Call the hardware function

- 3) Hardware function do the computation
- 4) Sent back the result to the called function

HLS: Square of number



HLS: Square of number



```
1 #include<stdio.h>
2 #include "head.h"
3
4 void square_hw(hls::stream<axis_data> &input, hls::stream<axis_data> & output);
5 void square_sw(int input[SIZE], int output[SIZE])
6 {
7     for(int i=0;i<SIZE;i++)
8     {
9         output[i]=input[i]*input[i];
10    }
11 }
12
```

HLS: Square of number

Vivado HLS 2019.1 - sq_of_no_array (D:\Vivado_lab\IIT_Kotyam\SQ_OF_NO\HLS\sq_of_no_array)

File Edit Project Solution Window Help

Synthesis(solution1)(square_hw_csynth.rpt) square_tb.cpp Debug Synthesis Analysis

```
1 #include<stdio.h>
2 #include "head.h"
3
4 void square_hw(hls::stream<axis_data> &input, hls::stream<axis_data> & output);
5 void square_sw(int input[SIZE], int output[SIZE])
6 {
7     for(int i=0;i<SIZE;i++)
8     {
9         output[i]=input[i]*input[i];
10    }
11 }
12
13 int main()
14 {
15
16     //input will be same for software and hardware function
17     int input[SIZE],output_sw[SIZE],output_hw[SIZE];           //two output one fo
18 }
```

Writable Smart Insert 9 : 37

HLS: Square of number

The screenshot shows the Vivado HLS 2019.1 interface. The title bar reads "Vivado HLS 2019.1 - sq_of_no_array (D:\Vivado_lab\IIT_Kotyam\SQ_OF_NO\HLS\sq_of_no_array)". The menu bar includes File, Edit, Project, Solution, Window, Help. The toolbar has various icons for file operations. The top right features tabs for Debug, Synthesis, and Analysis, with Synthesis selected. The main workspace displays the following C++ code:

```
13 int main()
14 {
15
16     //input will be same for software and hardware function
17     int input[SIZE],output_sw[SIZE],output_hw[SIZE];           //two output one fo
18     for(int i=0;i<SIZE;i++)
19     {
20         input[i]=i;                //populating the input
21     }
22     square_sw(input,output_sw);        //calling the software function
23
24     hls::stream<axis_data> in_vec,out_vec;
25     axis_data local_read,local_write;
26
27     for(int i=0;i<SIZE;i++)
28     {
29         local_read.data=input[i];      //storing input into data of struct
```

The code implements a square calculation. It first initializes an array 'input' with values from 0 to SIZE-1. It then calls a software function 'square_sw' which takes 'input' and 'output_sw' as parameters. Finally, it uses an HLS stream to read from 'input' and write to 'output_sw'. The code is annotated with comments explaining each step.

HLS: Square of number

Vivado HLS 2019.1 - sq_of_no_array (D:\Vivado_lab\IIIT_Kotyam\SQ_OF_NO\HLS\sq_of_no_array)

File Edit Project Solution Window Help

Synthesis(solution1)(square_hw_csynth.rpt) square_tb.cpp Debug Synthesis Analysis

```
27     for(int i=0;i<SIZE;i++)
28     {
29         local_read.data=input[i];           //storing input into data of struct
30         if(i==SIZE-1)
31         {
32             local_read.last=1;           //storing input into last of struct
33         }
34         else
35         {
36             local_read.last=0;           //storing input into last of struct
37         }
38         in_vec.write(local_read);       //writing the value of struct into input
39     }
40     square_hw(in_vec,out_vec);          // calling the hardware function
41
42
43     for(int i=0;i<SIZE;i++)
44     {
```

Writable Smart Insert 9 : 37

HLS: Square of number

Vivado HLS 2019.1 - sq_of_no_array (D:\Vivado_lab\IIIT_Kotyam\SQ_OF_NO\HLS\sq_of_no_array)

File Edit Project Solution Window Help

Synthesis(solution1)(square_hw_csynth.rpt) square_tb.cpp

Debug Synthesis Analysis

```
40     square_hw(in_vec,out_vec);           // calling the hardware function
41
42
43     for(int i=0;i<SIZE;i++)
44     {
45         local_write=out_vec.read();          //storing the data from stream to p
46         if(output_sw[i]!=local_write.data)   //comparing the ouptut from software
47         {
48             printf("Error at %d",i);
49             return 1;
50         }
51     }
52     printf("No error\n");
53     return 0;
54
55 }
```

Writable Smart Insert 9 : 37

HLS: Square of number

Vivado HLS 2019.1 - sq_of_no_array (D:\Vivado_lab\IIT_Kotyam\SQ_OF_NO\HLS\sq_of_no_array)

File Edit Project Solution Window Help

Synthesis(solution1)(square_hw_csynth.rpt) square_tb.cpp square.cpp

Debug Synthesis Analysis

```
1 #include "head.h"
2
3 void square_hw(hls::stream<axis_data> &input, hls::stream<axis_data> & output)
4 {
5 #pragma HLS INTERFACE axis register both port=output           //ports of the IP
6 #pragma HLS INTERFACE axis register both port=input           //ports of the IP
7 #pragma HLS INTERFACE ap_ctrl_none port=return               //To only consider last signal and discard other signals
8
9     axis_data local_read,local_write;
10    for(int i=0;i<SIZE;i++)
11    {
12        local_read=input.read();
13        local_write.data=(local_read.data)*(local_read.data);
14        if(i==SIZE-1)
15        {
16            local_write.last=1;
17        }
18        else
19        {
20            local_write.last=0;
21        }
22        output.write(local_write);
23    }
24 }
```

Writable Smart Insert 17 : 10

HLS: Square of number

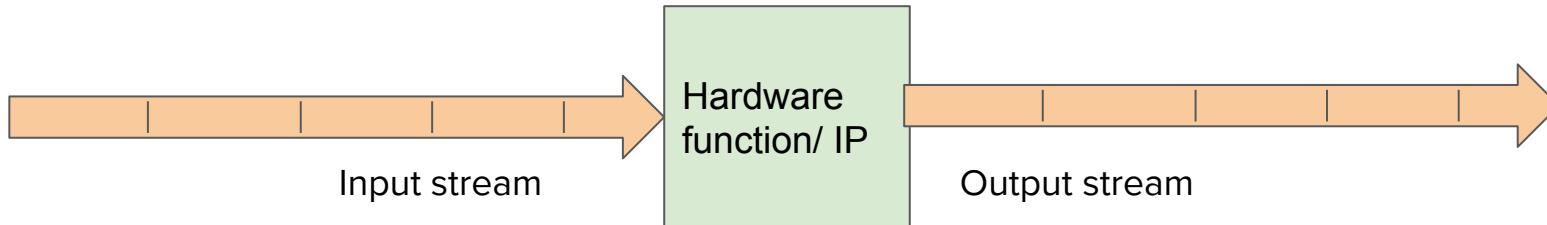
The screenshot shows the Vivado HLS 2019.1 interface. The title bar reads "Vivado HLS 2019.1 - sq_of_no_array (D:\Vivado_lab\IIT_Kotyam\SQ_OF_NO\HLS\sq_of_no_array)". The menu bar includes File, Edit, Project, Solution, Window, Help. The toolbar has various icons for file operations. The tabs at the top are Synthesis(solution1)(square_hw_csynth.rpt), square_tb.cpp, square.cpp, and head.h. The status bar at the bottom shows Writable, Smart Insert, and 1 : 1.

```
1 #include <hls_stream.h>
2 #define SIZE 1024    // //It is used for hls::stream
3 #include <ap_int.h> //It is used for defining even
4
5
6 struct axis_data{
7     ap_uint<32> data;
8     ap_uint<1> last;
9
10 };
11
```

HLS : Stream size

```
struct axis_data{  
    ap_uint<32> data;  
    ap_uint<1> last;  
};
```

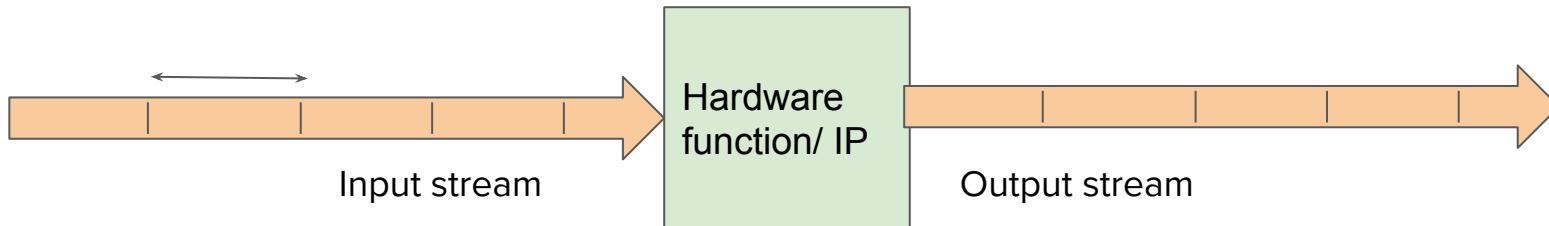
- The maximum amount of data can be transferred in 1 iteration.
- Stream works in FIFO manner



HLS : Stream size

```
struct axis_data{  
    ap_uint<32> data;  
    ap_uint<1> last;  
};
```

- The maximum amount of data can be transferred in 1 iteration.
- Stream works in FIFO manner



HLS: Square of number

File Edit Project Solution Window Help

Synthesis(solution1)(square_hw_csynth.rpt) square_tb.cpp head.h square.cpp

Debug Synthesis Analysis

```
1 #include<stdio.h>
2 #include "head.h"
3
4 void square_hw(hls::stream<axis_data> &input, hls::stream<axis_data> & output);
5 void square_sw(int input[SIZE], int output[SIZE])
6 {
7     for(int i=0;i<SIZE;i++)
8     {
9         output[i]=input[i]*input[i];
10    }
11 }
12
13 int main()
14 {
15
16     //input will be same for software and hardware function
17     int input[SIZE],output_sw[SIZE],output_hw[SIZE];           //two output one for s/w and other for h/w
18     for(int i=0;i<SIZE;i++)
19     {
20         input[i]=i;           //populating the input
21     }
22     square_sw(input,output_sw);           //calling the software function
23
24     hls::stream<axis_data> in_vec,out_vec;
25     axis_data local_read,local_write;
26
27     for(int i=0;i<SIZE;i++)
28     {
29         local_read.data=input[i];           //storing input into data of struct
30         if(i==SIZE-1)
31         {
32             local_read.last=1;           //storing input into last of struct
33         }
34         else
35         {
36             local_read.last=0;           //storing input into last of struct
37         }
38         in_vec=>local_write;
39     }
40     output_hw=>local_write;
41 }
```

Writable Smart Insert 42 : 1

HLS: Square of number

Vivado HLS 2019.1 - sq_of_no_array (D:\Vivado_lab\IIT_Kotyam\SQ_OF_NO\HLS\sq_of_no_array)

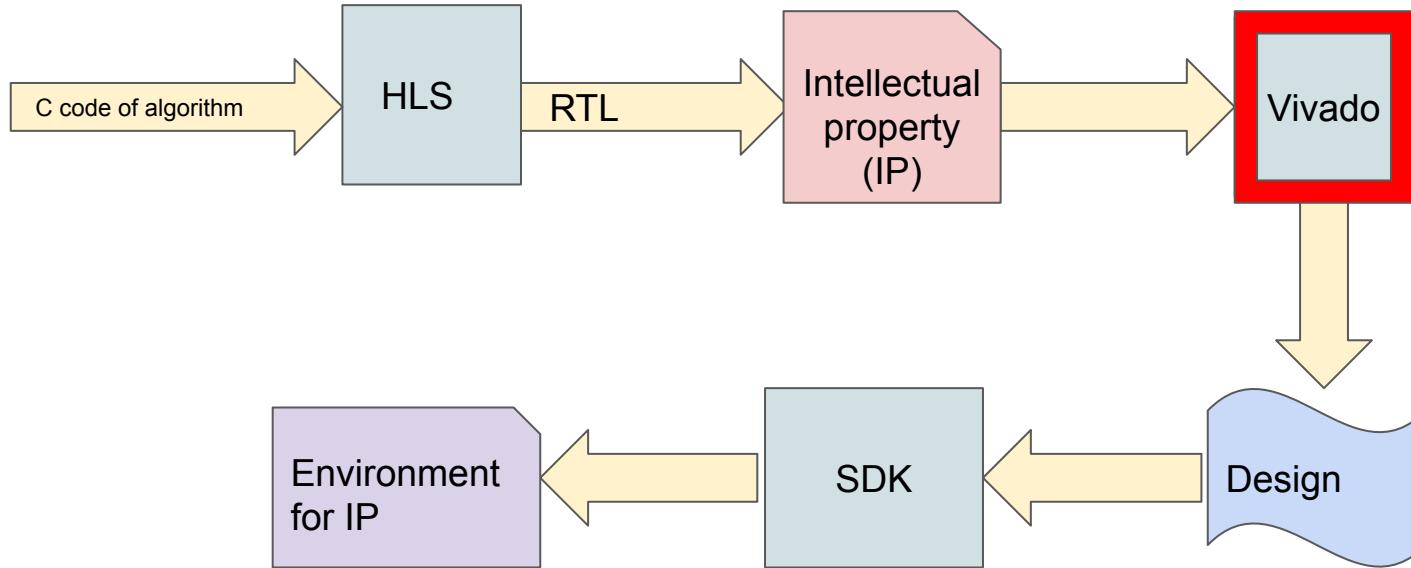
File Edit Project Solution Window Help

Synthesis(solution1)(square_hw_csynth.rpt) square_tb.cpp head.h square.cpp Debug Synthesis Analysis

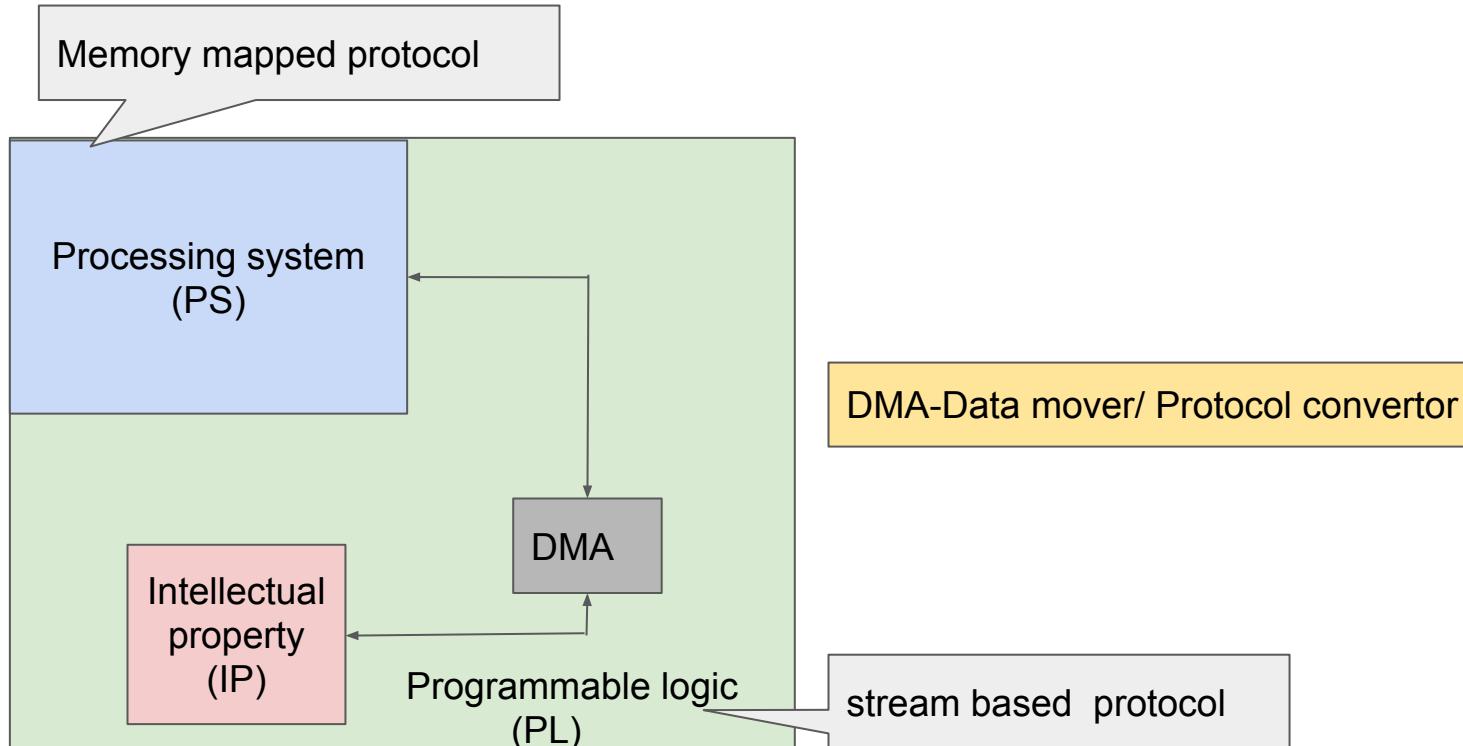
```
22     square_sw(input,output_sw);      //calling the software function
23
24     hls::stream<axis_data> in_vec,out_vec;
25     axis_data local_read,local_write;
26
27     for(int i=0;i<SIZE;i++)
28     {
29         local_read.data=input[i];          //storing input into data of struct
30         if(i==SIZE-1)
31         {
32             local_read.last=1;            //storing input into last of struct
33         }
34         else
35         {
36             local_read.last=0;            //storing input into last of struct
37         }
38         in_vec.write(local_read);        //writing the value of struct into input stream
39     }
40     square_hw(in_vec,out_vec);        // calling the hardware function
41
42
43     for(int i=0;i<SIZE;i++)
44     {
45         local_write=out_vec.read();      //storing the data from stream to pre-define variable
46         if(output_sw[i]!=local_write.data) //comparing the output from software and hardware function
47         {
48             printf("Error at %d",i);
49             return 1;
50         }
51     }
52     printf("No error\n");
53     return 0;
54 }
55
56
```

Writable Smart Insert 42 : 1

FPGA: Workflow

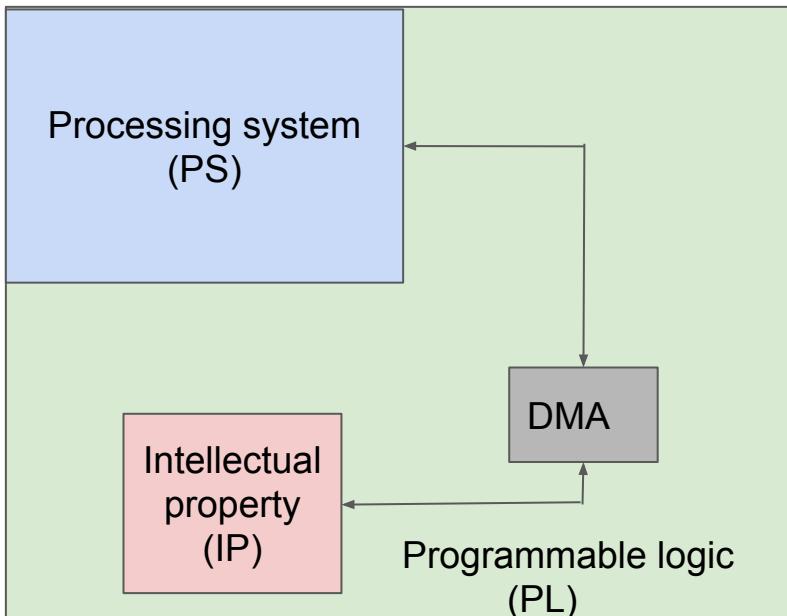


FPGA end-to-end working



*DMA- Direct memory access

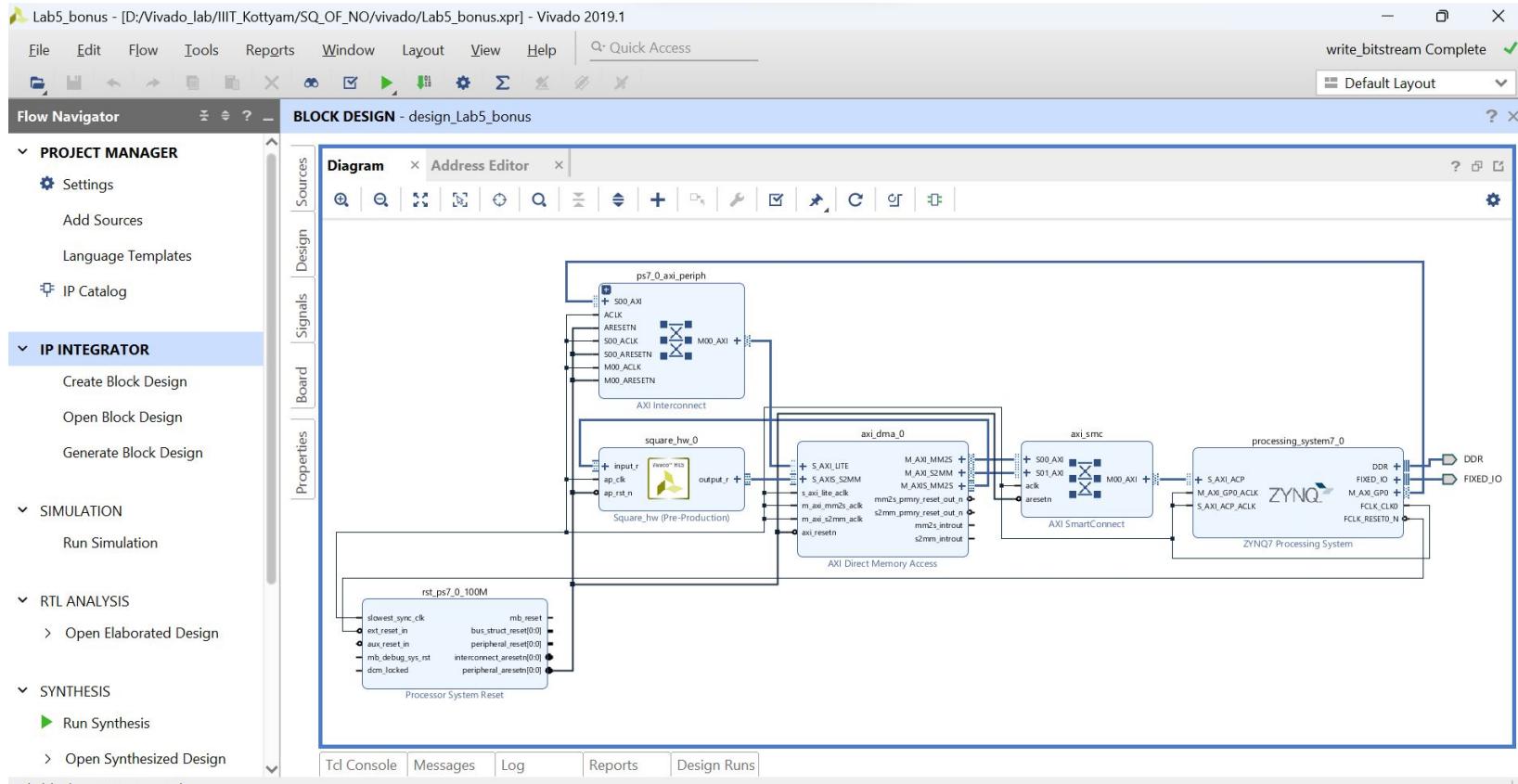
FPGA end-to-end working



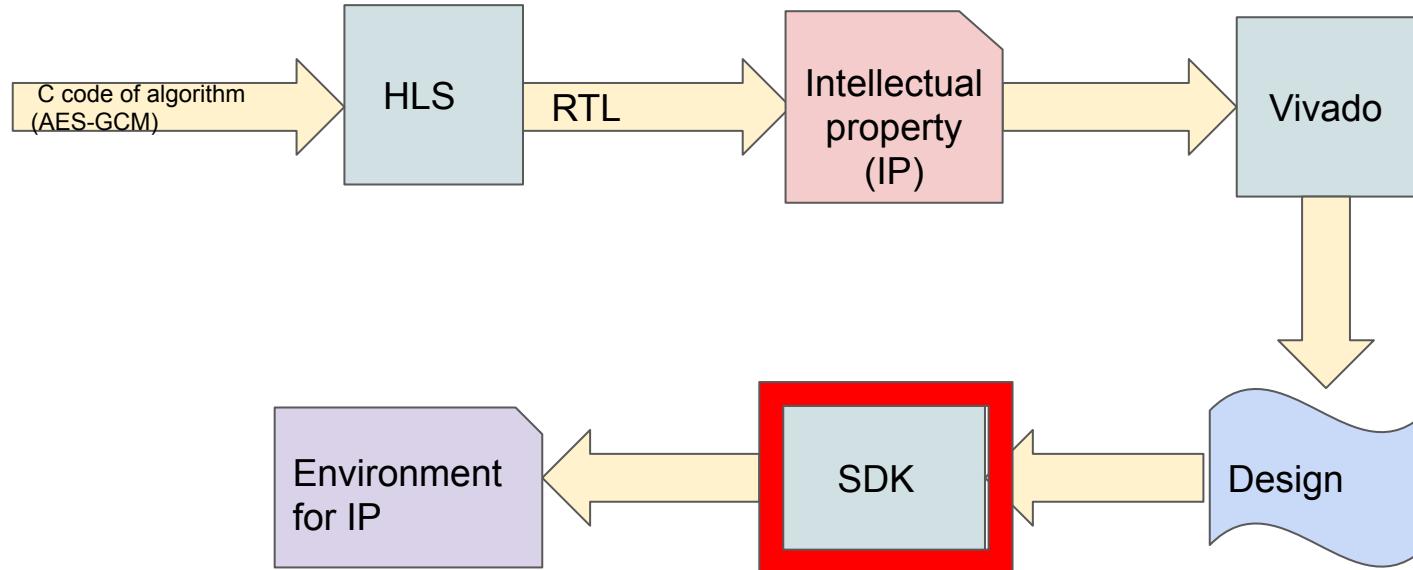
- 1) Master and slave
- 2) Master is always initiator of the transaction.
- 3) Transaction can be read/write

*DMA- Direct memory access

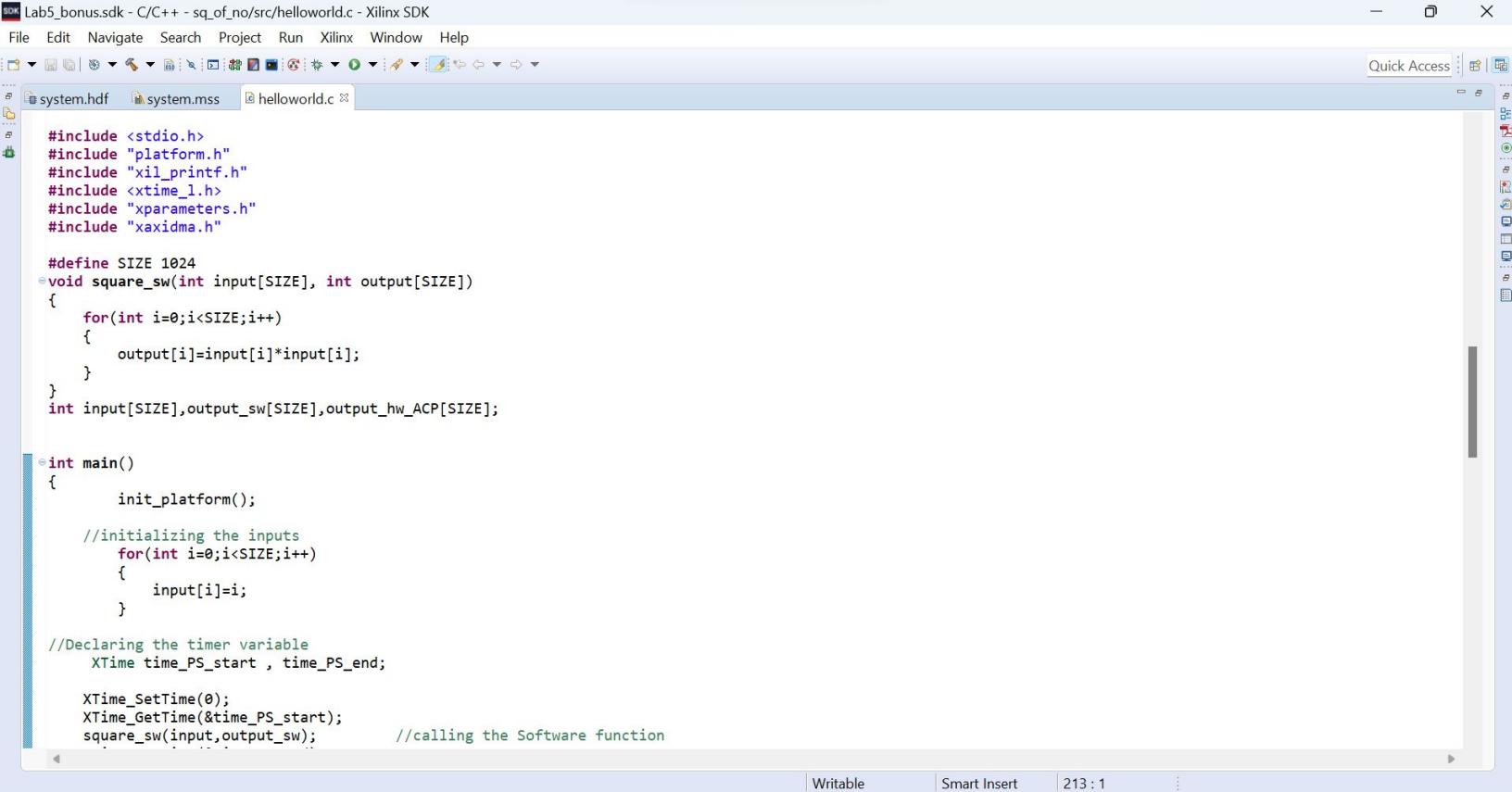
Vivado : Creation of block diagram



FPGA: Workflow



SDK



Lab5_bonus.sdk - C/C++ - sq_of_no/src/helloworld.c - Xilinx SDK

File Edit Navigate Search Project Run Xilinx Window Help

Quick Access

system.hdf system.mss helloworld.c

```
#include <stdio.h>
#include "platform.h"
#include "xil_printf.h"
#include <xtime.h>
#include "xparameters.h"
#include "xaxidma.h"

#define SIZE 1024
void square_sw(int input[SIZE], int output[SIZE])
{
    for(int i=0;i<SIZE;i++)
    {
        output[i]=input[i]*input[i];
    }
}
int input[SIZE],output_sw[SIZE],output_hw_ACP[SIZE];

int main()
{
    init_platform();

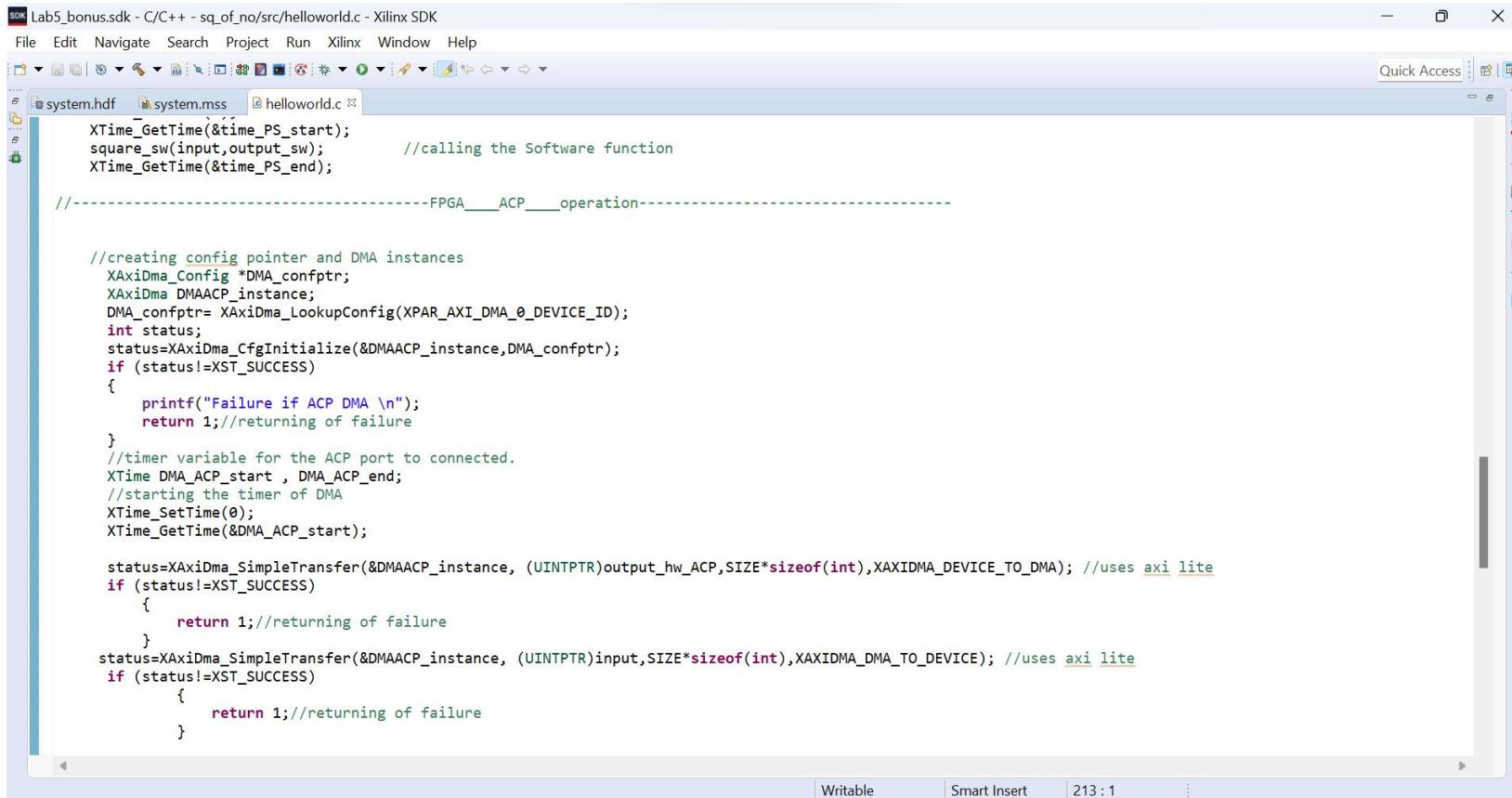
    //initializing the inputs
    for(int i=0;i<SIZE;i++)
    {
        input[i]=i;
    }

    //Declaring the timer variable
    XTime time_PS_start , time_PS_end;

    XTime_SetTime(0);
    XTime_GetTime(&time_PS_start);
    square_sw(input,output_sw);           //calling the Software function
}
```

Writable Smart Insert 213 : 1

SDK



The screenshot shows the Xilinx SDK IDE interface with the project "Lab5_bonus.sdk" open. The file "helloworld.c" is the active editor. The code implements a software function "square_sw" and performs DMA operations using AXI DMA and ACP ports.

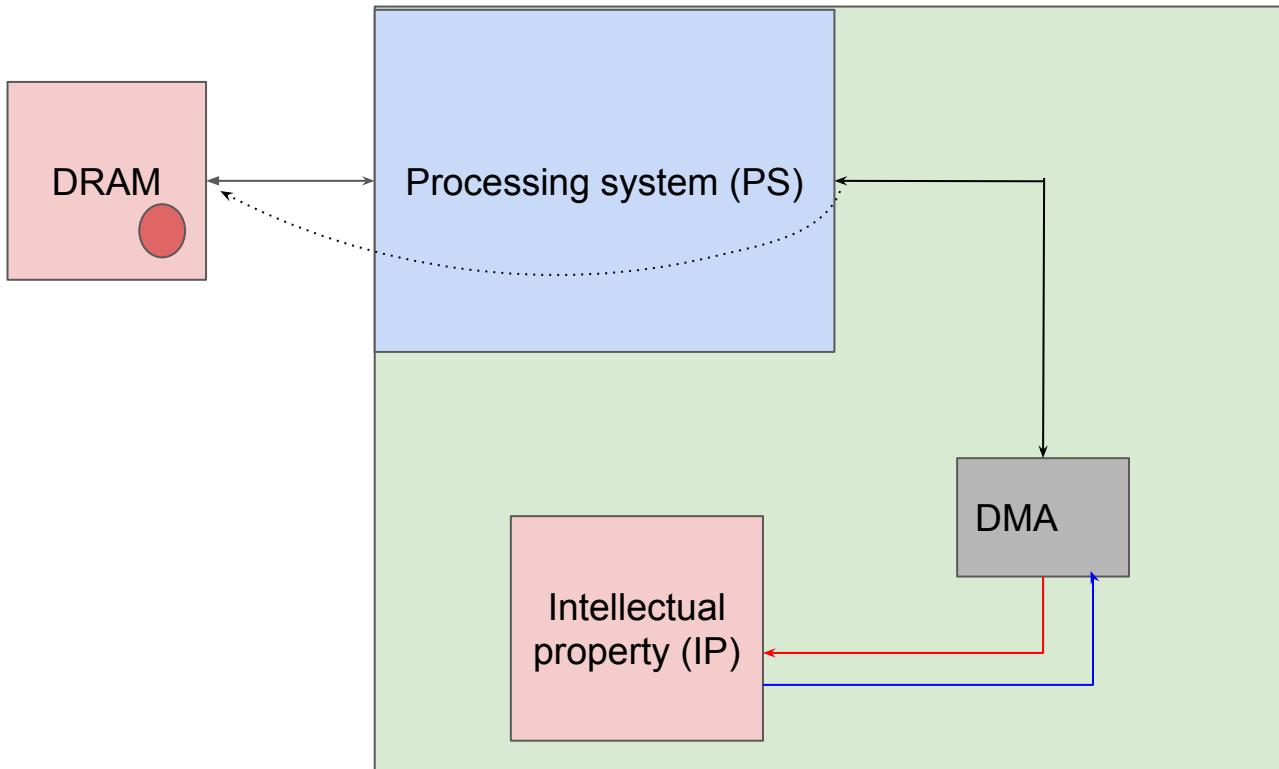
```
Lab5_bonus.sdk - C/C++ - sq_of_no/src/helloworld.c - Xilinx SDK
File Edit Navigate Search Project Run Xilinx Window Help
system.hdf system.mss helloworld.c
XTIME_GetTime(&time_PS_start);
square_sw(input,output_sw); //calling the Software function
XTIME_GetTime(&time_PS_end);

//-----FPGA ACP operation-----

//creating config pointer and DMA instances
XAxiDma_Config *DMA_confptr;
XAxiDma DMAACP_instance;
DMA_confptr= XAxiDma_LookupConfig(XPAR_AXI_DMA_0_DEVICE_ID);
int status;
status=XAxiDma_CfgInitialize(&DMAACP_instance,DMA_confptr);
if (status!=XST_SUCCESS)
{
    printf("Failure if ACP DMA \n");
    return 1;//returning of failure
}
//timer variable for the ACP port to connected.
XTIME DMA_ACP_start , DMA_ACP_end;
//starting the timer of DMA
XTIME_SetTime(0);
XTIME_GetTime(&DMA_ACP_start);

status=XAxiDma_SimpleTransfer(&DMAACP_instance, (UINTPTR)output_hw_ACP,SIZE*sizeof(int),XAXIDMA_DEVICE_TO_DMA); //uses axi lite
if (status!=XST_SUCCESS)
{
    return 1;//returning of failure
}
status=XAxiDma_SimpleTransfer(&DMAACP_instance, (UINTPTR)input,SIZE*sizeof(int),XAXIDMA_DMA_TO_DEVICE); //uses axi lite
if (status!=XST_SUCCESS)
{
    return 1;//returning of failure
}
```

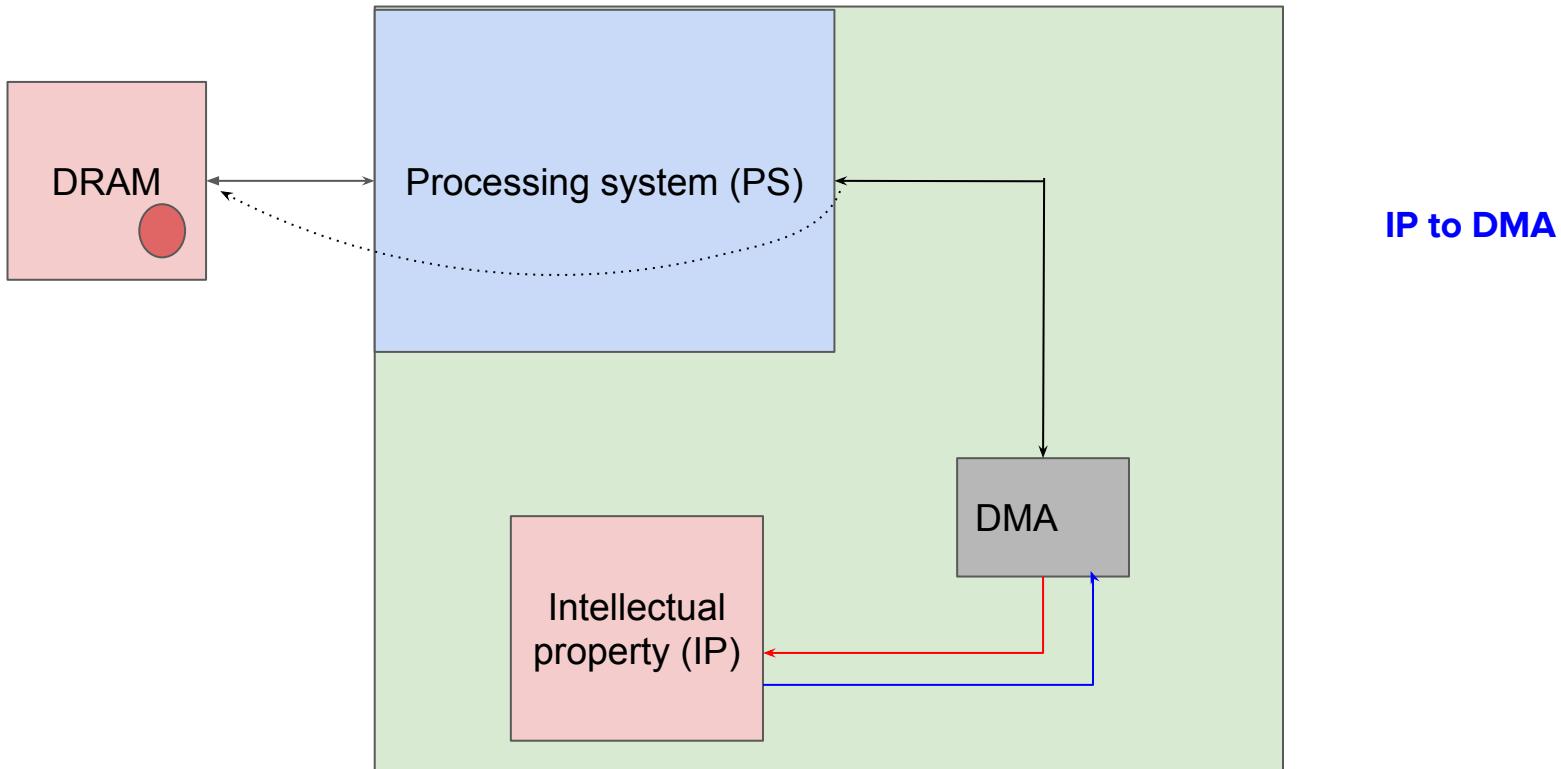
Link from DMA to IP and IP to DMA



Which link to setup first?

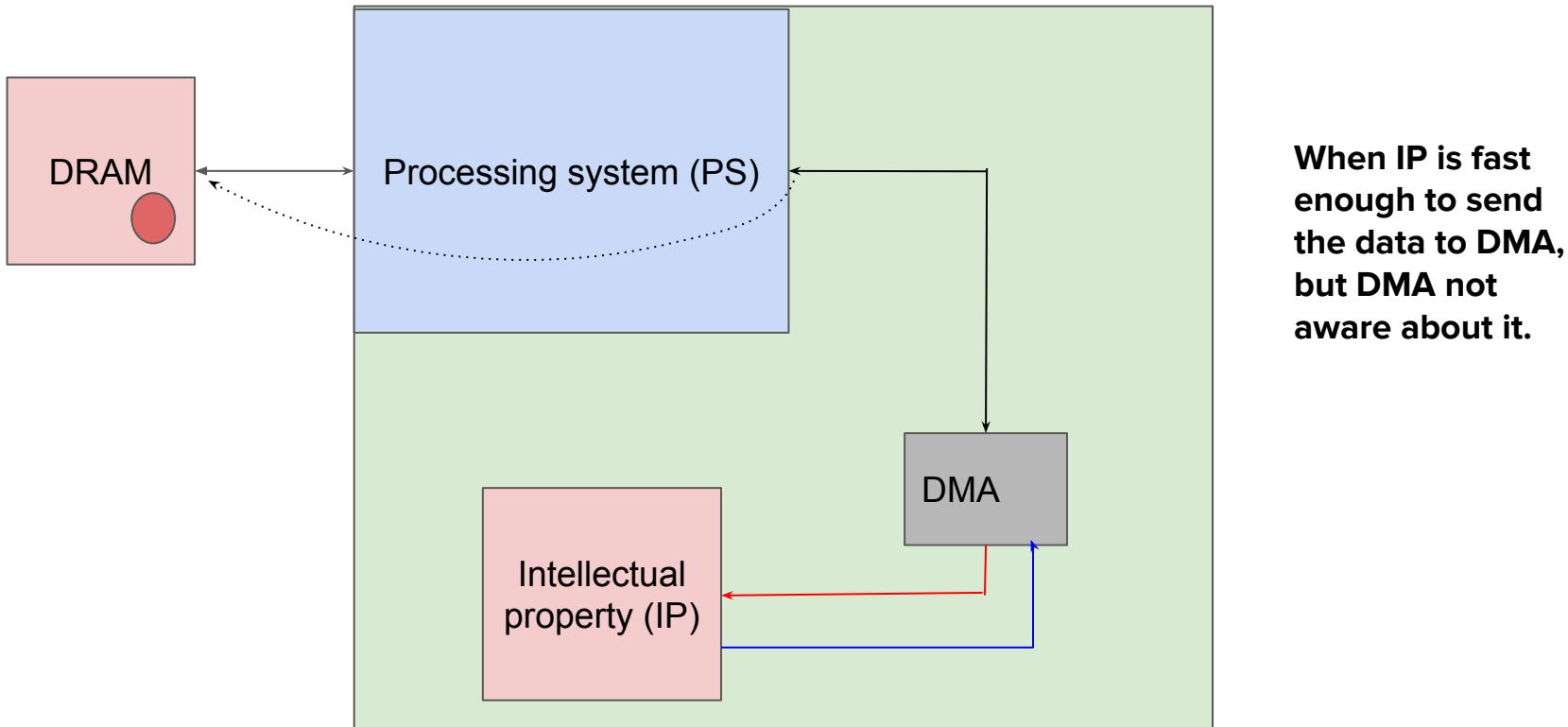
*DMA- Direct memory access

Link from DMA to IP and IP to DMA



*DMA- Direct memory access

Back pressure



*DMA- Direct memory access

SDK

Lab5_bonus.sdk - C/C++ - sq_of_no/src/helloworld.c - Xilinx SDK

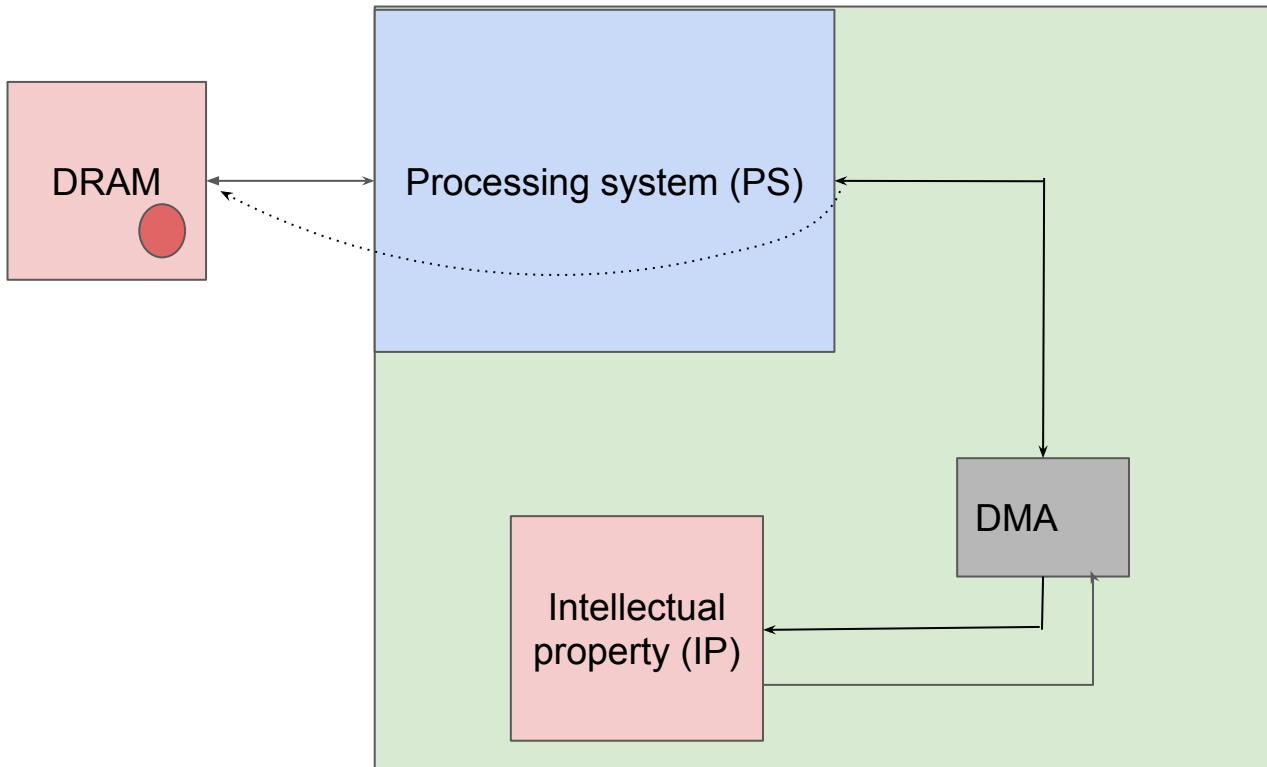
File Edit Navigate Search Project Run Xilinx Window Help

system.hdf system.mss helloworld.c

```
{  
    return 1;//returning of failure  
}  
  
//    XAxiDMA_readreg(Base add. of dma)  
status=XAxiDma_ReadReg(XPAR_AXI_DMA_0_BASEADDR,0x04) & 0x00000002;  
//    status=status&0x00000002;  
while(status!=0x00000002)  
{  
    status=XAxiDma_ReadReg(XPAR_AXI_DMA_0_BASEADDR,0x04) & 0x00000002;  
}  
status=XAxiDma_ReadReg(XPAR_AXI_DMA_0_BASEADDR,0x34) & 0x00000002;  
while(status!=0x00000002)  
{  
    status=XAxiDma_ReadReg(XPAR_AXI_DMA_0_BASEADDR,0x34) & 0x00000002;  
}  
XTIME_GetTime(&DMA_AC_end);  
  
for(int i=0;i<SIZE;i++)  
{  
    if(output_sw[i]!=output_hw_ACP[i])  
    {  
        printf("Error at %d",i);  
        return 1;  
    }  
}  
printf("No error\n");  
  
printf("\n PS and PL output: \r\n");  
for (int i = 0 ; i < SIZE ; i++)  
{  
    printf("output for PS: %d \n",output_sw[i] );  
    printf("output for PL: %d \n",output_hw_ACP[i] );  
}
```

Writable Smart Insert 213 : 1

Polling : to check is DMA idle?



*DMA- Direct memory access

SDK

Lab5_bonus.sdk - C/C++ - sq_of_no/src/helloworld.c - Xilinx SDK

File Edit Navigate Search Project Run Xilinx Window Help

Quick Access

```
system.hdf system.mss helloworld.c
if(output_sw[i]!=output_hw_ACP[i])
{
    printf("Error at %d",i);
    return 1;
}
printf("No error\n");

printf("\n PS and PL output: \r\n");
for (int i = 0 ; i < SIZE ; i++)
{
    printf("output for PS: %d \n",output_sw[i] );
    printf("output for PL: %d \n",output_hw_ACP[i] );
}

printf("\n-----EXECUTION TIME of PL-----\n");
float time_processorrr = 0;
time_processorrr = (float)1.0 * (DMA_ACP_end - DMA_ACP_start) / (COUNTS_PER_SECOND/1000000);
printf("Execution Time for PL in Micro-Seconds : %f\n" , time_processorrr);

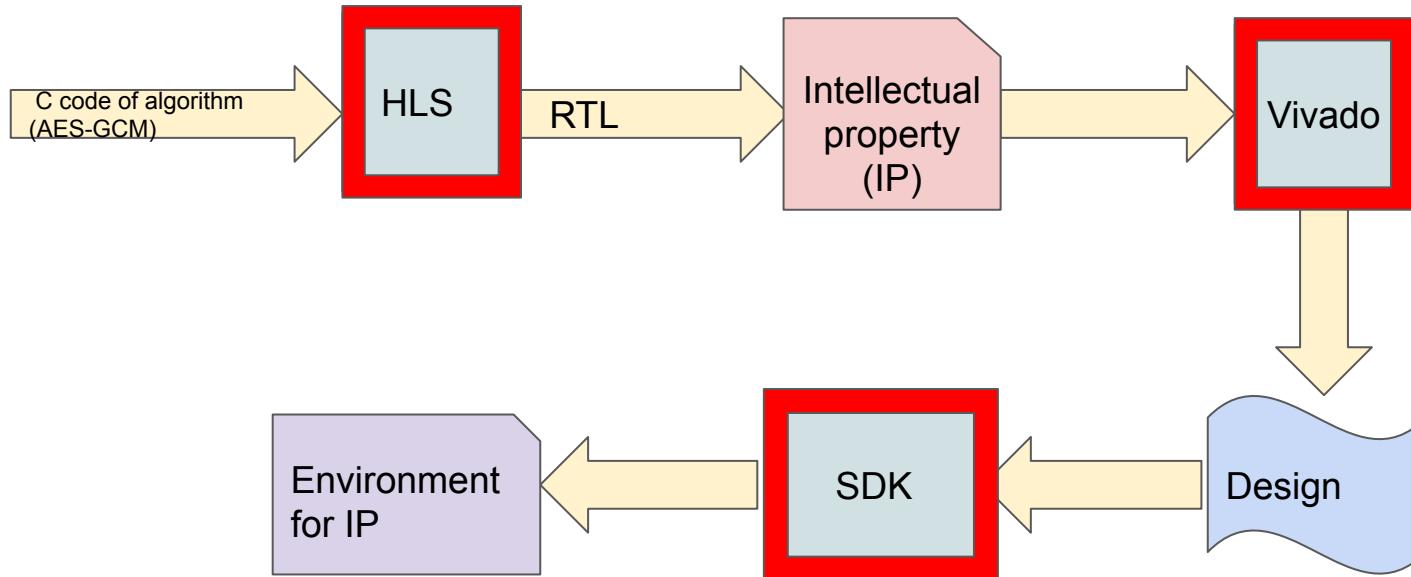
//PS time
printf("\n-----EXECUTION TIME OF PS-----\n");
float time_processor = 0;
time_processor = (float)1.0 * (time_PS_end - time_PS_start) / (COUNTS_PER_SECOND/1000000);
printf("Execution Time for PS in Micro-Seconds : %f\n" , time_processor);
cleanup_platform();
return 0;
}
```

Writable Smart Insert 213 : 1

Setting up DMA

- Lookup configuration
- Initializing configuration
- Setting up the link/channel from DMA to IP and IP to DMA
- Keep polling, the status of channel

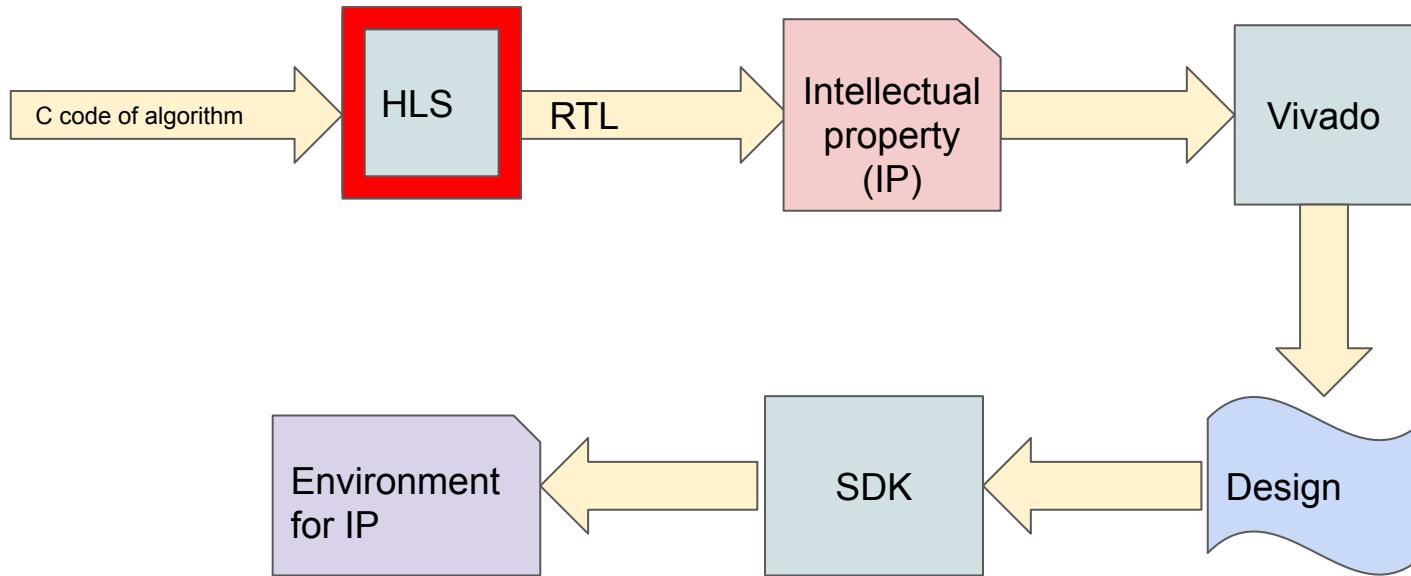
Recap



FPGA hands-on

- Matrix multiplication program

Recap : Workflow



Matrix multiplication program

The screenshot shows a software development environment with the following interface elements:

- Menu Bar:** File, Edit, Project, Solution, Window, Help.
- Toolbar:** Standard icons for file operations like Open, Save, Print, etc.
- Explorer View:** Shows the project structure under "Lab6.1".
 - Includes
 - Source
 - mat.h
 - matrix.cpp
 - Test Bench
 - matrix_tb.cpp
 - solution1
 - constraints
 - directives.tcl
 - script.tcl
 - csim
 - build
 - report
 - impl
 - ip
 - misc
 - verilog
 - vhdl
 - sim
 - autowrap
 - report
 - tv
 - verilog
 - wrapc
 - wrapc_pc
- Editor View:** Displays the code for "matrix_tb.cpp".

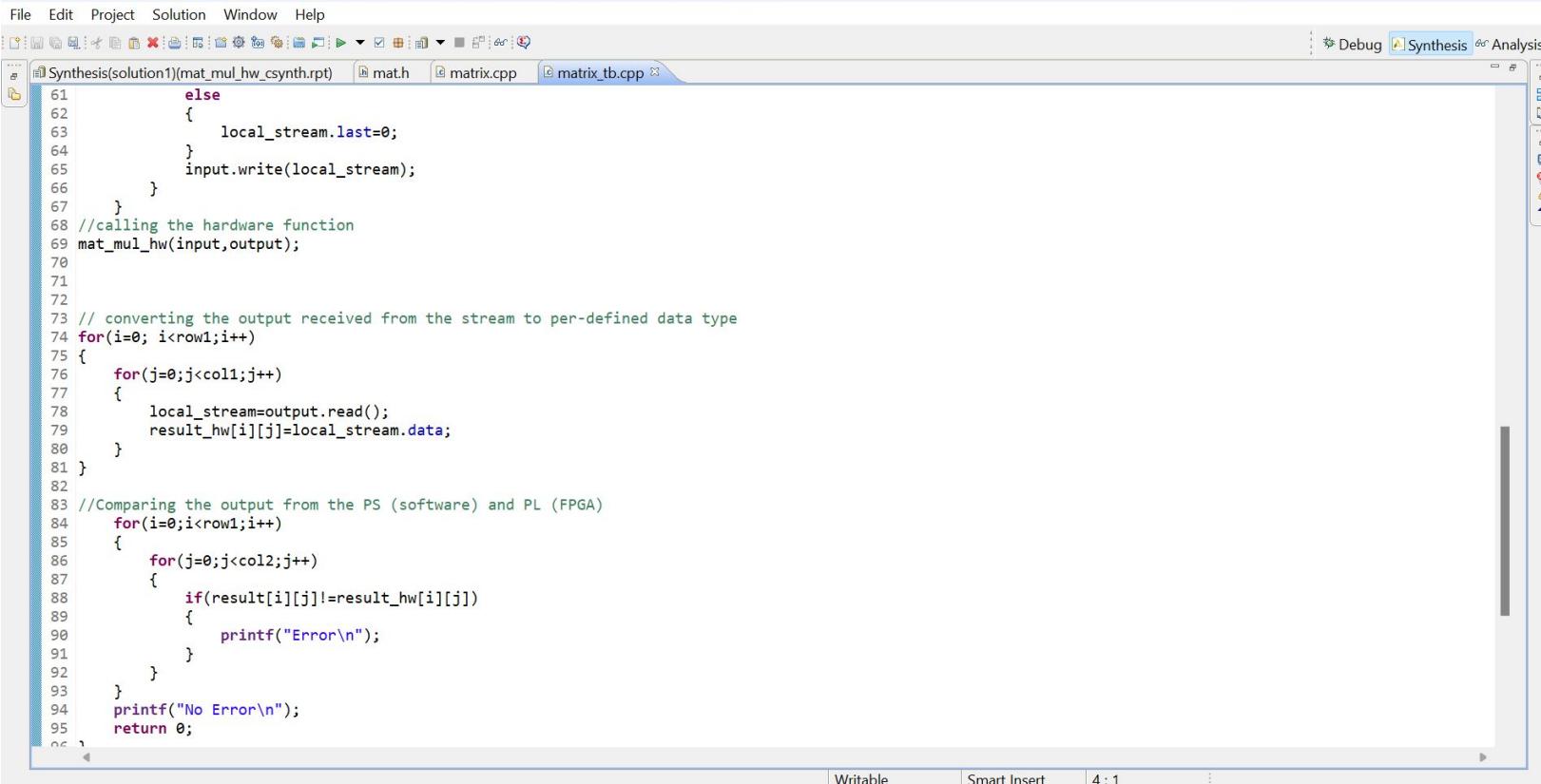
```
1 #include "math.h"
2
3 void mat_mul(M_TYPE mat1[row1][col1], M_TYPE mat2[row2][col2], M_TYPE result[row1][col2]);
4
5 int main()
6 {
7     M_TYPE mat1[row1][col1], mat2[row2][col2], result[row1][col2], result_hw[row1][col2];
8     int i, j, k;
9
10    //populating the input
11    for(i=0; i<row1; i++)
12    {
13        for(j=0; j<col1; j++)
14            mat1[i][j]=i;
15    }
16
17    for(i=0; i<row2; i++)
18    {
19        for(j=0; j<col2; j++)
20            mat2[i][j]=i;
21    }
22
23
24    //calling software function of matrix mul.
25    mat_mul(mat1, mat2, result);
26
27 //-----Hardware function-----
28 hls::stream<axis_data> input, output;
29 axis_data local_stream;
30
31 //Generating the input for stream from pre-defined data types
32 for(i=0; i<row1; i++)
```
- Toolbars:** Debug, Synthesis.

Matrix multiplication program

The screenshot shows the Vivado HLS IDE interface. The top menu bar includes File, Edit, Project, Solution, Window, and Help. The tabs at the top right are Debug, Synthesis, and Analysis. The left sidebar is the Explorer view, showing a project structure for 'Lab6.1' with sub-folders like Includes, Source (containing mat.h and matrix.cpp), Test Bench (matrix_tb.cpp), solution1, constraints, csim, impl, sim, and syn. The main editor window displays C++ code for generating input streams for matrix multiplication. The bottom navigation bar includes tabs for Console, Errors, Warnings, DRCs, and a Vivado HLS Console tab. The status bar at the bottom shows Writable, Smart Insert, and a page number of 4 : 1.

```
31 //Generating the input for stream from pre-defined data types
32 for(i=0;i<row1;i++)
33 {
34     for(j=0;j<col1;j++)
35     {
36         local_stream.data=mat1[i][j];
37         //Generating the last signal
38         if(i==2 && j==2)
39         {
40             local_stream.last=1;
41         }
42         else
43         {
44             local_stream.last=0;
45         }
46         input.write(local_stream);
47     }
48 }
49
50
51 for(i=0;i<row2;i++)
52 {
53     for(j=0;j<col2;j++)
54     {
55         local_stream.data=mat2[i][j];
56
57         if(i==row2-1 && j==col2-1)
58         {
59             local_stream.last=1;
60         }
61         else
62     }
```

Matrix multiplication program



The screenshot shows a software development environment with a code editor displaying a C++ program. The window title is "Synthesis(solution1)(mat_mul_hw_csynth.rpt)". The menu bar includes File, Edit, Project, Solution, Window, Help. The toolbar has various icons for file operations. The code editor shows the following code:

```
File Edit Project Solution Window Help
Synthesis(solution1)(mat_mul_hw_csynth.rpt) mat.h matrix.cpp matrix_tb.cpp Debug Synthesis Analysis

61     else
62     {
63         local_stream.last=0;
64     }
65     input.write(local_stream);
66 }
67 }
68 //calling the hardware function
69 mat_mul_hw(input,output);
70
71
72
73 // converting the output received from the stream to per-defined data type
74 for(i=0; i<row1;i++)
75 {
76     for(j=0;j<col1;j++)
77     {
78         local_stream=output.read();
79         result_hw[i][j]=local_stream.data;
80     }
81 }
82
83 //Comparing the output from the PS (software) and PL (FPGA)
84 for(i=0;i<row1;i++)
85 {
86     for(j=0;j<col2;j++)
87     {
88         if(result[i][j]!=result_hw[i][j])
89         {
90             printf("Error\n");
91         }
92     }
93 }
94 printf("No Error\n");
95 return 0;
96 }
```

The status bar at the bottom indicates Writable, Smart Insert, and 4 : 1.

Matrix multiplication program

The screenshot shows the Vivado HLS IDE interface with the following components:

- File Edit Project Solution Window Help**: Standard menu bar.
- Explorer**: Shows the project structure under "solution1".
- Synthesis(solution1)(mat_mul_hw_csynth.rpt)**: Synthesis report tab.
- math.h**, **matrix.cpp**, **matrix_tb.cpp**: Source files tabs.
- Code Editor**: Displays the C++ code for matrix multiplication. The code uses nested loops to calculate the result matrix by summing products of corresponding row and column elements.
- Outliner**: Shows the hierarchical structure of the code, including **mat.h**, **mat_mul(M_TYPE[], M_TYPE[])**, and **main() : int**.
- Debug**, **Synthesis**, **Analysis**: Tool tabs.
- Console**: Shows the Vivado HLS Console output.
- Errors**, **Warnings**, **DRCs**: Status indicators.
- Writable**, **Smart Insert**, **104 : 29**: Text editor status bar.

```
85     for(j=0;j<col2;j++)
86     {
87         if(result[i][j]!=result_hw[i][j])
88         {
89             printf("Error\n");
90         }
91     }
92 }
93 printf("No Error\n");
94 return 0;
95 }

99 void mat_mul(M_TYPE mat1[row1][col1], M_TYPE mat2[row2][col2], M_TYPE result[row1][col2])
100 {
101     int i,j,k,sum;
102     for( i=0;i<row1;i++)
103     {
104         for( j=0;j<col2;j++)
105         {
106             sum=0;
107             for( k=0;k<row2;k++)
108             {
109                 sum+=(mat1[i][k]*mat2[k][j]);
110             }
111             result[i][j]=sum;
112         }
113     }
114     return ;
115 }
116 }
```

Matrix multiplication program

The screenshot shows the Vivado HLS IDE interface with the following components:

- File Edit Project Solution Window Help**: Standard menu bar.
- Explorer**: Shows the project structure under "solution1".
- Synthesis(solution1)(mat_mul_hw_csynth.rpt)**: Synthesis report tab.
- math.h**, **matrix.cpp**, **matrix_tb.cpp**: Source files tabs.
- Code Editor**: Displays the C++ code for matrix multiplication. The code uses nested loops to calculate the result matrix by summing products of corresponding row and column elements.
- Outliner**: Shows the hierarchical structure of the design, including **math.h**, **mat_mul(M_TYPE[], M_TYPE[])**, and **main() : int**.
- Debug**, **Synthesis**, **Analysis**: Tool tabs.
- Console**: Shows the Vivado HLS Console output.
- Errors**, **Warnings**, **DRCs**: Status indicators.
- Writable**, **Smart Insert**, **104 : 29**: Text editor status bar.

```
86     for(j=0;j<col2;j++)
87     {
88         if(result[i][j]!=result_hw[i][j])
89         {
90             printf("Error\n");
91         }
92     }
93     printf("No Error\n");
94     return 0;
95 }
96 }

99 void mat_mul(M_TYPE mat1[row1][col1], M_TYPE mat2[row2][col2], M_TYPE result[row1][col2])
100{
101    int i,j,k,sum;
102    for( i=0;i<row1;i++)
103    {
104        for( j=0;j<col2;j++)
105        {
106            sum=0;
107            for( k=0;k<row2;k++)
108            {
109                sum+=(mat1[i][k]*mat2[k][j]);
110            }
111            result[i][j]=sum;
112        }
113    }
114    return ;
115 }
116 }
```

Matrix multiplication program

The screenshot shows the Vivado HLS IDE interface with the following components:

- File Bar:** File, Edit, Project, Solution, Window, Help.
- Toolbars:** Standard icons for file operations like Open, Save, and Build.
- Explorer View:** Shows the project structure under "Lab6.1".
 - Includes
 - Source
 - math.h
 - matrix.cpp
 - Test Bench
 - matrix_tb.cpp
 - solution1
 - * constraints
 - directives.tcl
 - script.tcl
 - csim
 - build
 - report
 - impl
 - ip
 - misc
 - verilog
 - vhdl
 - sim
 - autowrap
 - report
 - tv
 - verilog
 - wrapc
 - wrapc_pc
 - syn
 - report
 - systemc
 - verilog
- Editor View:** Displays the C++ code for "matrix.cpp".

```
1 #include<stdio.h>
2 #include <hls_stream.h>
3 #include <ap_int.h>
4 typedef float M_TYPE;
5 //Matrix order i.e no. of row and col in input matrix
6 #define row1 8
7 #define col1 8
8 #define row2 8
9 #define col2 8
10
11 struct axis_data{
12     M_TYPE data;
13     ap_uint<1> last;
14 };
15
16 void mat_mul_hw(hls::stream<axis_data> &input, hls::stream<axis_data> &output);
17 |
18
19
```
- Synthesis View:** Shows the synthesized code and parameters.
 - stdio.h
 - hls_stream.h
 - ap_int.h
 - M_TYPE : float
 - # row1
 - # col1
 - # row2
 - # col2
 - > axis_data
 - < mat_mul_hw(hls::stream<
- Console View:** Vivado HLS Console tab.
- Status Bar:** Writable, Smart Insert, 17 : 1.

Matrix multiplication program

The screenshot shows the Vivado HLS IDE interface with the following details:

- File Menu:** File, Edit, Project, Solution, Window, Help.
- Toolbars:** Standard toolbar with icons for file operations, search, and help.
- Explorer View:** Shows the project structure under "Lab6.1".
 - Includes
 - Source
 - mat.h
 - matrix.cpp
 - Test Bench
 - matrix_tb.cpp
 - solution1
 - * constraints
 - directives.tcl
 - script.tcl
 - csim
 - build
 - report
 - impl
 - ip
 - misc
 - verilog
 - vhdl
 - sim
 - autowrap
 - report
 - tv
 - verilog
 - wrapc
 - wrapc_pc
 - syn
 - report
 - systemc
 - verilog
- Code Editor:** Displays the C++ code for "matrix.cpp".

```
#include "mat.h"
void mat_mul_hw(hls::stream<axis_data> & input, hls::stream<axis_data> &output)
{
#pragma HLS INTERFACE axis register both port=output
#pragma HLS INTERFACE axis register both port=input
#pragma HLS INTERFACE ap_ctrl_none port=return
M_TYPE mat1[row1][col1], mat2[row2][col2],mat_res[row1][col2];
int i, j, k, sum;
axis_data local_stream;
//Storing streaming data to respective data type variables
for( i=0;i<row1;i++)
{
    for( j=0;j<col1;j++)
    {
        local_stream=input.read();
        mat1[i][j]=local_stream.data;
    }
}
for( i=0;i<row2;i++)
{
    for( j=0;j<col2;j++)
    {
        local_stream=input.read();
        mat2[i][j]=local_stream.data;
    }
}
```
- Analysis Tab:** Shows "Synthesis(solution1)(mat_mul_hw.csynth.rpt)".
- Outliner:** Shows the hierarchical structure of the design.
 - math
 - mat_mul_hw(hls::stream<
- Console:** Vivado HLS Console tab.
- Bottom Bar:** Writable, Smart Insert, 16 : 10, and other interface controls.

Matrix multiplication program : Analysis

The screenshot shows the Xilinx Vivado IDE interface during the synthesis process. The top menu bar includes File, Edit, Project, Solution, Window, Help, Debug, Synthesis, and Analysis. The central workspace displays the 'Performance Estimates' and 'Utilization Estimates' reports for the 'solution1' project.

Performance Estimates

- Timing (ns)**
 - Summary**

Clock	Target	Estimated	Uncertainty
ap_dclk	10.00	7.303	1.25
 - Latency (clock cycles)**
 - Summary**

Latency	Interval			
min	max	min	max	Type
7556	7556	7556	7556	none
 - Detail**
 - Instance**
 - Loop**

Utilization Estimates

 - Summary**

Name	BRAM_18K	DSP48E	FF	LUT	URAM
DSP	-	-	-	-	-
Expression	-	-	0	826	-
FIFO	-	-	-	-	-
Instance	-	5	688	1265	-
Memory	3	-	0	0	0
Multiplexer	-	-	-	381	-
Register	-	-	491	-	-
Total	3	5	1179	2472	0
Available	280	220	106400	53200	0
Utilization (%)	1	2	1	4	0

Matrix multiplication program

The screenshot shows a software development environment with the following interface elements:

- File Edit Project Solution Window Help**: The top menu bar.
- Explorer**: A tree view of project files under "Lab6.1".
- Synthesis(solution1)(mat_mul_hw_hlsynth.rpt)**: The current synthesis report.
- math.h**, **matrix.cpp**, **matrix_tb.cpp**: The source files for the project.
- Debug Synthesis Analysis**: The active synthesis tab.
- Outliner**: A panel showing the file structure.
- Code Editor**: The main window displaying the C++ code for matrix multiplication.

```
30     }
31 }
32
33 //Matrix Multiplication computation
34
35     for( i=0;i<row1;i++)
36     {
37         for( j=0;j<col2;j++)
38         {
39             sum=0;
40             for( k=0;k<row2;k++)
41             {
42                 sum+=mat1[i][k]*mat2[k][j];
43             }
44             mat_res[i][j]=sum;
45         }
46     }
47
48 //writing back result to the stream
49     for( i=0;i<row1;i++)
50     {
51         for( j=0;j<col2;j++)
52         {
53             local_stream.data=mat_res[i][j];
54             if((row1-1==i) &&((col1-1==j)))
55                 local_stream.last=1;
56             else
57                 local_stream.last=0;
58             output.write(local_stream);
59         }
60     }
61
62 }
```

Matrix multiplication program : Optimised

The screenshot shows a software development environment with the following interface elements:

- File Edit Project Solution Window Help**: The top menu bar.
- Explorer**: A tree view of project files under "lab6.3".
- Synthesis(solution1)(mat_mul_hw_2_csynth.rpt)**: The current file being edited in the main editor area.
- matrix_tb.cpp mat.h matrix.cpp**: Other files in the project.
- Debug Synthesis Analysis**: The tabs at the top right of the editor.
- Outli Direc**: The right-hand panel showing the synthesis results for "mat.h" and "mat_mul_hw_2(hls::stream<axis_data>)".

The code in the editor is as follows:

```
1 #include "mat.h"
2
3
4 void mat_mul_hw_2(hls::stream<axis_data> & input, hls::stream<axis_data> &output)
5 {
6     #pragma HLS INTERFACE axis register both port=output
7     #pragma HLS INTERFACE axis register both port=input
8     #pragma HLS INTERFACE ap_ctrl_none port=return
9
10    int i, j, k, sum;
11    M_TYPE mat1[row1][col1], mat2[row2][col2], mat_res[row1][col2];
12    #pragma HLS ARRAY_PARTITION variable=mat2 complete dim=2
13    #pragma HLS ARRAY_PARTITION variable=mat1 complete dim=1
14
15    axis_data local_stream;
16
17 //Storing streaming data to respective data type variables
18
19    for( i=0;i<row1;i++)
20    {
21        for( j=0;j<col1;j++)
22        {
23            #pragma HLS PIPELINE
24            local_stream=input.read();
25            mat1[i][j]=local_stream.data;
26        }
27    }
28
29    for( i=0;i<row2;i++)
30    {
31        for( j=0;j<col2;j++)
32        {
33            #pragma HLS PIPELINE
34            local_stream=input.read();
35            mat2[i][j]=local_stream.data;
36        }
37    }
38
39    for( i=0;i<row1;i++)
40    {
41        for( j=0;j<col2;j++)
42        {
43            #pragma HLS PIPELINE
44            local_stream=output.write();
45            mat_res[i][j]=sum;
46        }
47    }
48
49    #pragma HLS INTERFACE ap_ctrl_none port=return
50 }
```

Matrix multiplication program : Optimised

The screenshot shows the Vivado IDE interface with the following components:

- File Edit Project Solution Window Help**: The top menu bar.
- Explorer**: Shows the project structure under "solution1".
- Synthesis(solution1)(mat_mul_hw_2_csynth.rpt)**: The current synthesis report tab.
- matrix_tb.cpp mat.h matrix.cpp**: The source files for the project.
- Code Editor**: Displays the C++ code for matrix multiplication, annotated with HLS pragmas for pipeline optimization.
- Output**: Shows the synthesis results for "mat.h" and "mat_mul_hw_2(hls::stream)".
- Analysis**: A tab in the output panel.

```
36         }
37     }
38
39 //Matrix Multiplication computation
40
41     for( i=0;i<row1;i++)
42     {
43         for( j=0;j<col2;j++)
44         {
45             #pragma HLS PIPELINE
46
47                 sum=0;
48                 for( k=0;k<row2;k++)
49                 {
50                     sum+=mat1[i][k]*mat2[k][j];
51                 }
52                 mat_res[i][j]=sum;
53             }
54         }
55
56 //writing back result to the stream
57     for( i=0;i<row1;i++)
58     {
59         for( j=0;j<col2;j++)
60         {
61             #pragma HLS PIPELINE
62                 local_stream.data=mat_res[i][j];
63                 if((row1-1==i) &&((col1-1==j)))
64                     local_stream.last=1;
65                 else
66                     local_stream.last=0;
67                 output.write(local_stream);
68         }
69     }
70 }
```

Matrix multiplication Optimised: Analysis

The screenshot shows a software interface for analyzing a synthesized design. The top menu bar includes File, Edit, Project, Solution, Window, and Help. The left sidebar displays a project structure under 'solution1' with various sub-directories like 'Includes', 'Source', 'Test Bench', and 'constraints'. The main window contains two main sections: 'Performance Estimates' and 'Utilization Estimates'.

Performance Estimates

- Timing (ns)**
 - Summary**

Clock	Target	Estimated	Uncertainty
ap_clk	10.00	8.181	1.25
 - Latency (clock cycles)**
 - Summary**

Latency	Interval			
min	max	min	max	Type
377	377	377	377	none
 - Detail**
 - Instance**
 - Loop**

Utilization Estimates

 - Summary**

Name	BRAM_18K	DSP48E	FF	LUT	URAM
DSP	-	-	-	-	-
Expression	-	-	0	401	-
FIFO	-	-	-	-	-
Instance	-	40	5768	14504	-
Memory	1	-	1024	64	0
Multiplexer	-	-	-	1301	-
Register	0	-	5792	622	-
Total	1	40	12584	16892	0
Available	280	220	106400	53200	0
Utilization (%)	~0	18	11	31	0

Comparison between optimised and unoptimised

The screenshot shows the Vivado Design Suite interface with the following details:

- Performance Estimates:**
 - Timing (ns) Summary:** Clock: ap_clk, Target: 10.00, Estimated: 7.303, Uncertainty: 1.25.
 - Latency (clock cycles) Summary:** Latency: 7556, Interval: 7556, Type: none.
- Utilization Estimates:** A table showing resource usage across various components like DSP, Expression, FIFO, Instance, Memory, Multiplexer, Register, Total, Available, and Utilization (%). The utilization values are very low (mostly 0 or 1).

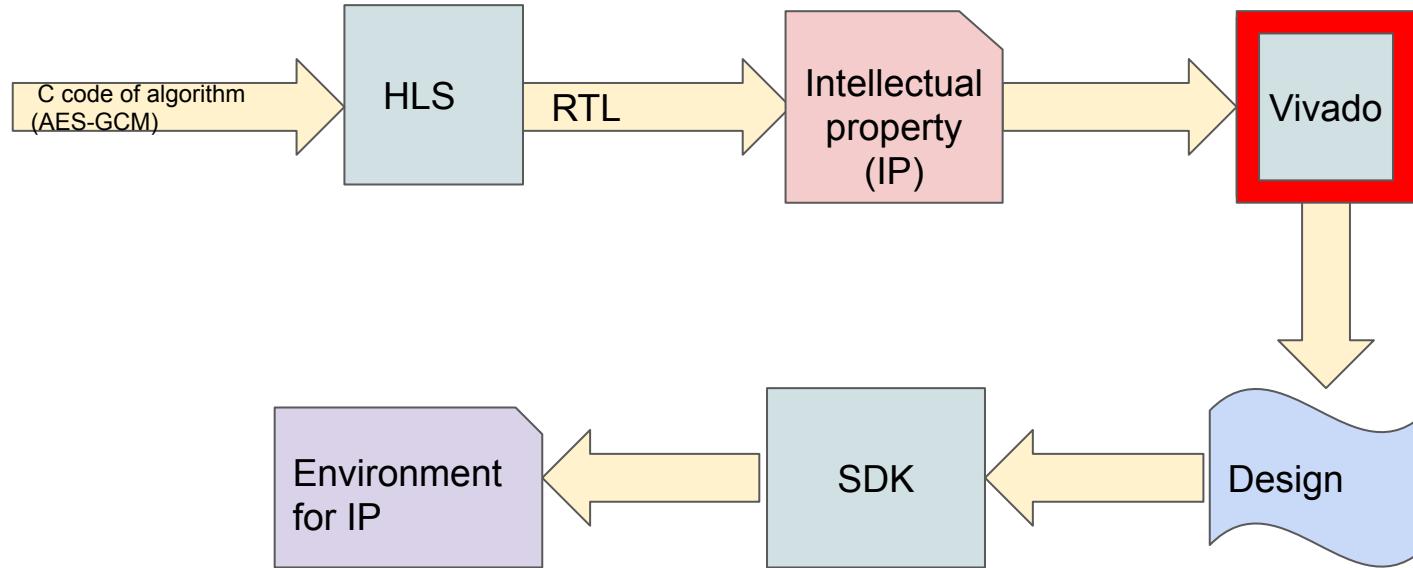
optimised

unoptimised

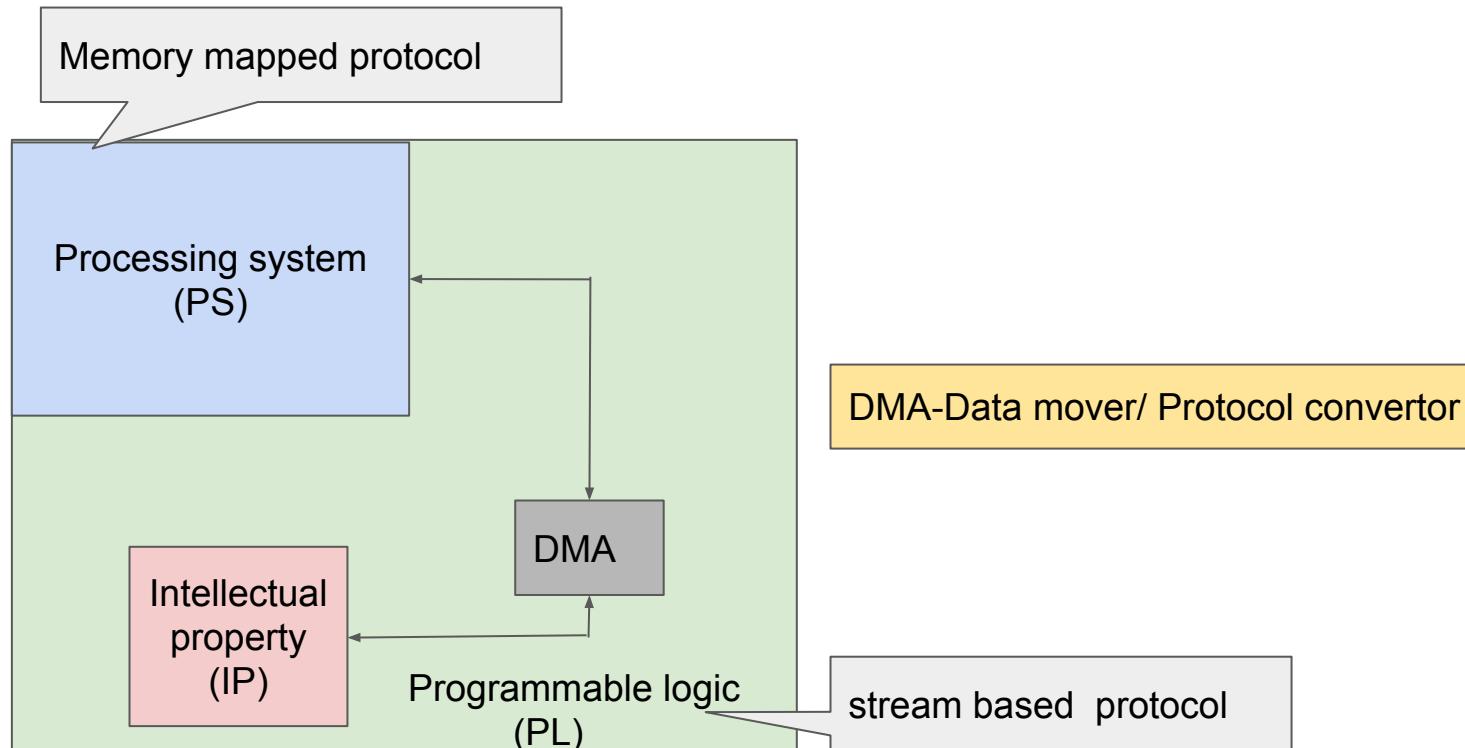
The screenshot shows the Vivado Design Suite interface with the following details:

- Performance Estimates:**
 - Timing (ns) Summary:** Clock: ap_clk, Target: 10.00, Estimated: 8.181, Uncertainty: 1.25.
 - Latency (clock cycles) Summary:** Latency: 377, Interval: 377, Type: none.
- Utilization Estimates:** A table showing resource usage across various components like DSP, Expression, FIFO, Instance, Memory, Multiplexer, Register, Total, Available, and Utilization (%). The utilization values are higher than in the unoptimized case, particularly for memory and registers.

FPGA: Workflow

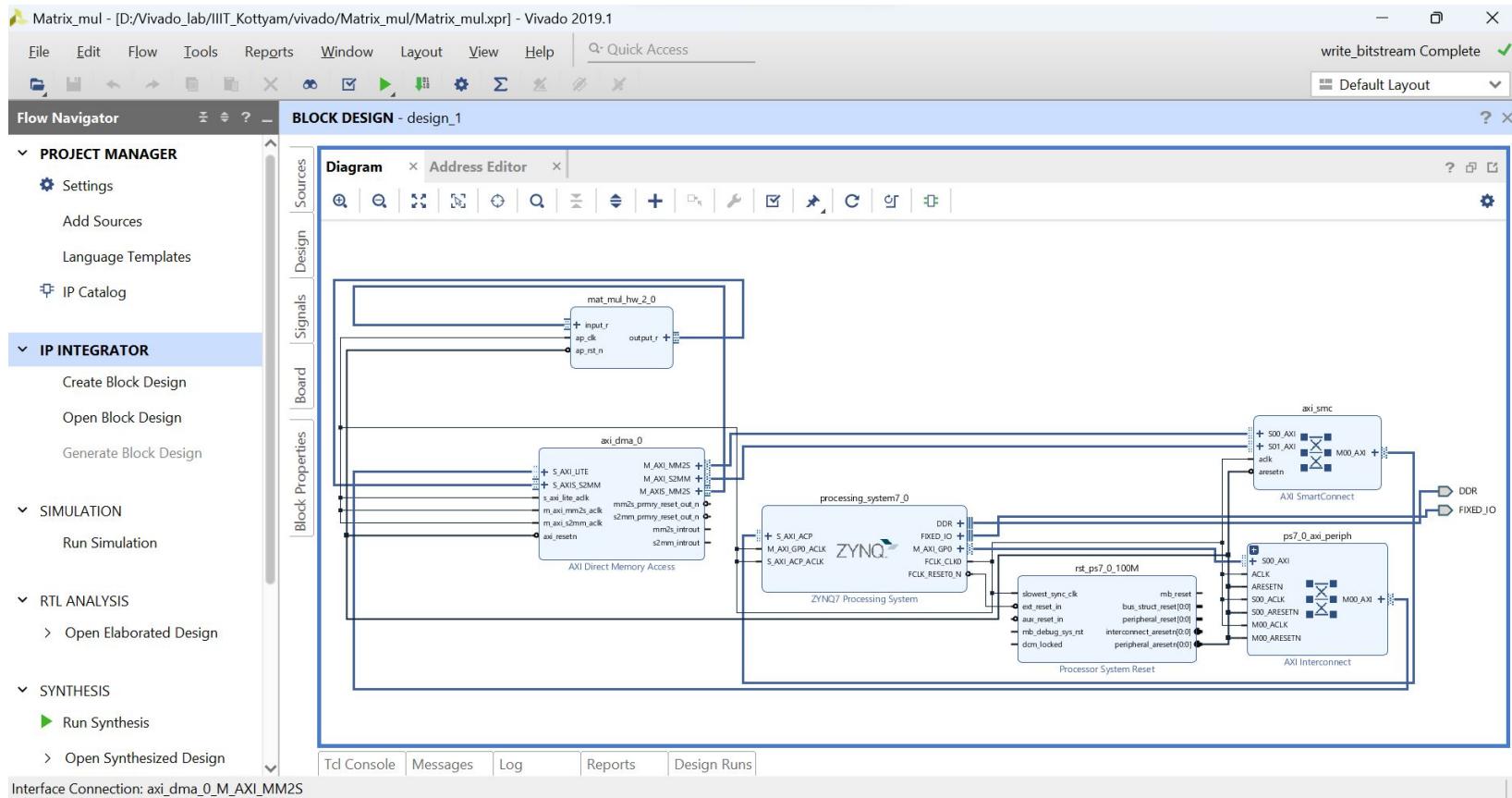


FPGA end-to-end working

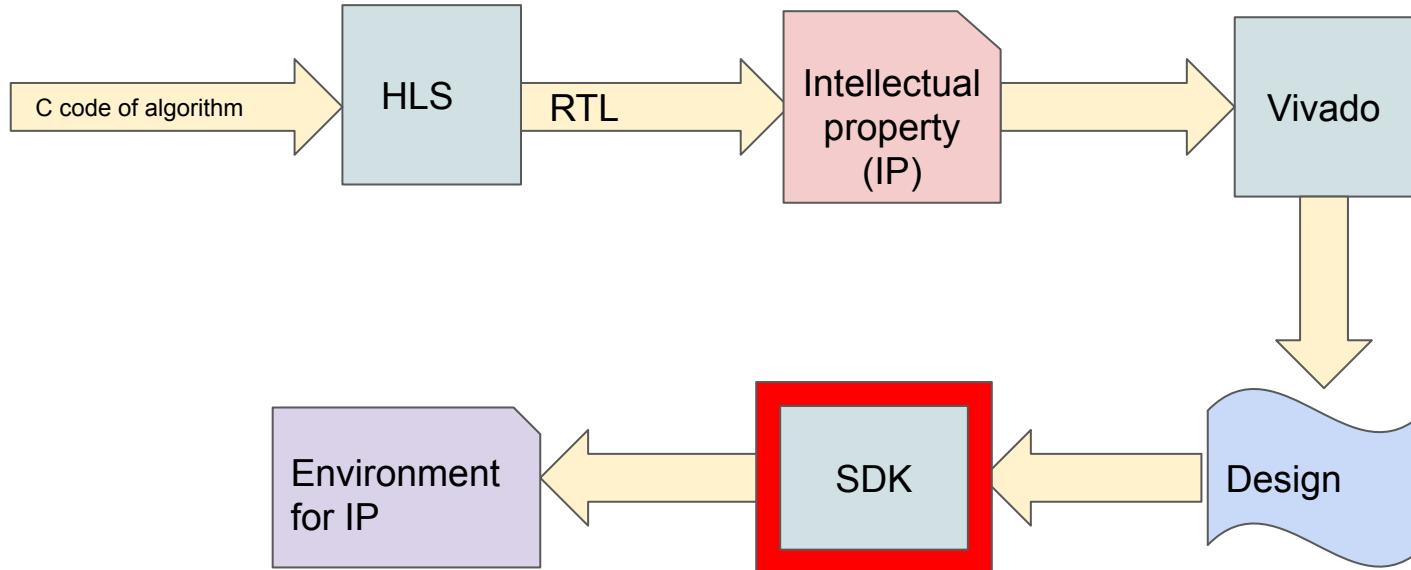


*DMA- Direct memory access

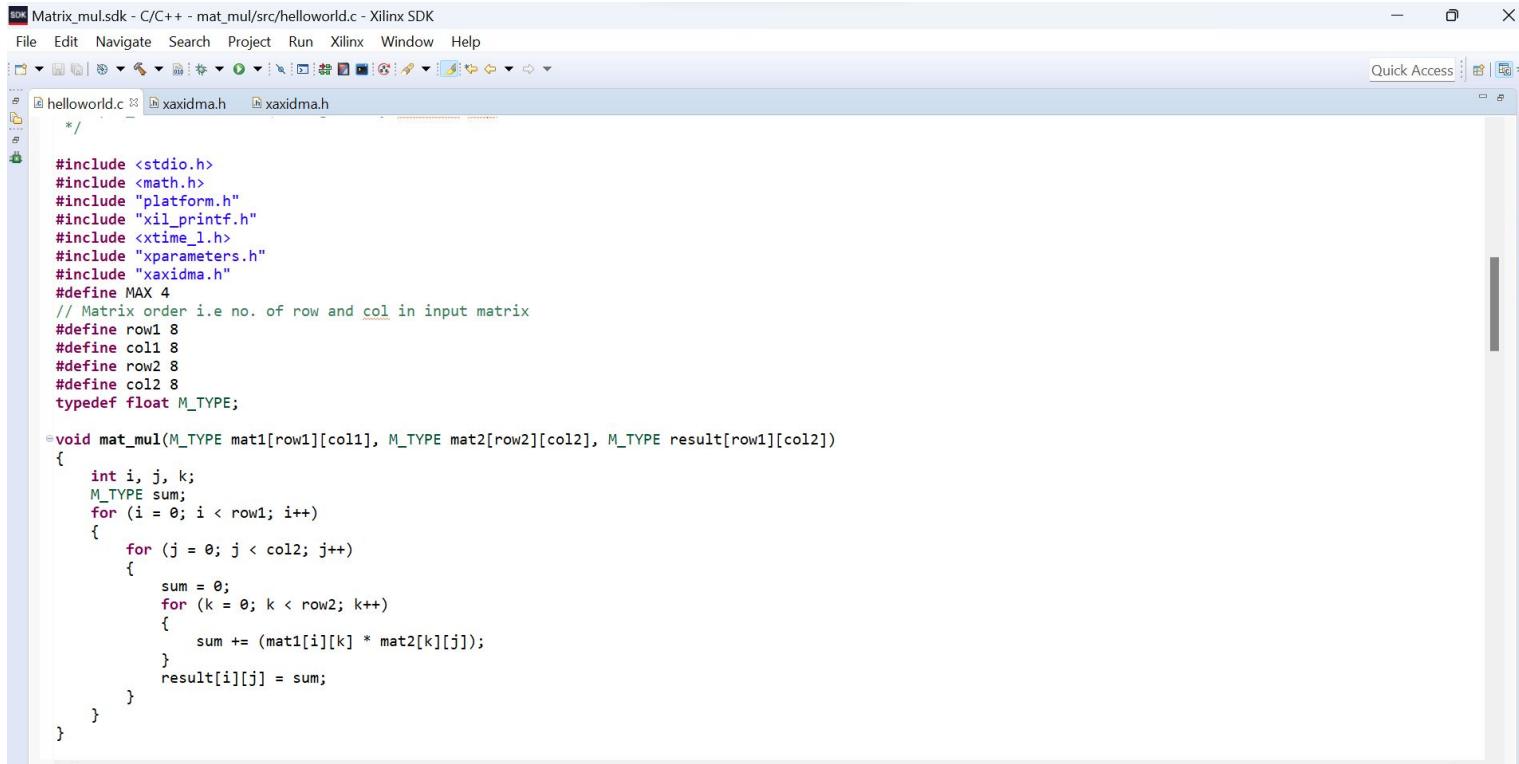
Block diagram: Vivado



FPGA: Workflow



SDK



The screenshot shows the Xilinx SDK IDE interface. The title bar reads "Matrix_mul.sdk - C/C++ - mat_mul/src/helloworld.c - Xilinx SDK". The menu bar includes File, Edit, Navigate, Search, Project, Run, Xilinx, Window, and Help. The toolbar has various icons for file operations like Open, Save, and Build. The Quick Access ribbon is visible on the right. The main workspace shows two files: "helloworld.c" and "xaxidma.h". The code editor displays the following C code:

```
#include <stdio.h>
#include <math.h>
#include "platform.h"
#include "xil_printf.h"
#include <xtime_l.h>
#include "xparameters.h"
#include "xaxidma.h"
#define MAX 4
// Matrix order i.e no. of row and col in input matrix
#define row1 8
#define col1 8
#define row2 8
#define col2 8
typedef float M_TYPE;

void mat_mul(M_TYPE mat1[row1][col1], M_TYPE mat2[row2][col2], M_TYPE result[row1][col2])
{
    int i, j, k;
    M_TYPE sum;
    for (i = 0; i < row1; i++)
    {
        for (j = 0; j < col2; j++)
        {
            sum = 0;
            for (k = 0; k < row2; k++)
            {
                sum += (mat1[i][k] * mat2[k][j]);
            }
            result[i][j] = sum;
        }
    }
}
```

SDK

Matrix_mul.sdk - C/C++ - mat_mul/src/helloworld.c - Xilinx SDK

File Edit Navigate Search Project Run Xilinx Window Help

Quick Access

```
helloworld.c x xaxidma.h xaxidma.h

}

int main()
{
    init_platform();

    //-----program_starting-----
    float sum_processor_PL_0 = 0, sum_processor_PL_1 = 0;
    float sum_processor_PS_0 = 0;
    int index=0;
    for (int a = 1; a <= MAX; a++)
    {
        M_TYPE mat1[row1][col1], mat2[row2][col2], result[row1][col2], result_hw[row1][col2];
        M_TYPE intermediate[index];
        int i, j;

        XTime seed_value;
        XTime_GetTime(&seed_value);
        srand(seed_value);

        // populating the input
        for (i = 0; i < row1; i++)
        {
            for (j = 0; j < col1; j++)
                mat1[i][j] = ((float)rand()/(RAND_MAX/6));
        }

        for (i = 0; i < row2; i++)
        {
            for (j = 0; j < col2; j++)
                mat2[i][j] = ((float)rand()/(RAND_MAX/6));
        }

        // Declaring the timer variable
    }
}
```

SDK

Matrix_mul.sdk - C/C++ - mat_mul/src/helloworld.c - Xilinx SDK

File Edit Navigate Search Project Run Xilinx Window Help

Quick Access

```
// Declaring the timer variable
XTIME time_PS_start, time_PS_end;
XTIME_SetTime(0);
XTIME_GetTime(&time_PS_start);
mat_mul(mat1, mat2, result); // calling software function
XTIME_GetTime(&time_PS_end);

for (i = 0; i < row1; i++)
{
    for (j = 0; j < col1; j++)
    {
        intermediate[index] = mat1[i][j];
        index=index+1;
    } // multiply by first 8 index
}
for (i = 0; i < row2; i++)
{
    for (j = 0; j < col2; j++)
    {
        intermediate[index] = mat2[i][j];
        index=index+1;
    }
}

//-----DMA_1_in_action-----//

// creating config pointer and DMA instances
XAXIDMA_Config *DMA_Confptr;
XAXIDMA DMAACP_instance;

DMA_Confptr = XAXIDma_LookupConfig(XPAR_AXI_DMA_0_DEVICE_ID);

int status;
status = XAXIDma_CfgInitialize(&DMAACP_instance, DMA_Confptr);
```

SDK

Matrix_mul.sdk - C/C++ - mat_mul/src/helloworld.c - Xilinx SDK

File Edit Navigate Search Project Run Xilinx Window Help

Quick Access

```
//-----DMA_1_in_action-----//  
  
// creating config pointer and DMA instances  
XAxiDma_Config *DMA_confptr;  
XAxiDma DMAACP_instance;  
  
DMA_confptr = XAxiDma_LookupConfig(XPAR_AXI_DMA_0_DEVICE_ID);  
  
int status;  
status = XAxiDma_CfgInitialize(&DMAACP_instance, DMA_confptr);  
if (status != XST_SUCCESS)  
{  
    printf("Failure if ACP DMA \n");  
    return 1; // returning of failure  
}  
  
// timer variable for the ACP port to connected.  
XTime DMA_ACP_start, DMA_ACP_end;  
// starting the timer of DMA  
XTime_SetTime(0);  
XTime_GetTime(&DMA_ACP_start);  
status = XAxiDma_SimpleTransfer(&DMAACP_instance, (UINTPTR)result_hw, row1 * col2 * sizeof(M_TYPE), XAXIDMA_DEVICE_TO_DMA); // uses axi lite  
if (status != XST_SUCCESS)  
{  
    return 1; // returning of failure  
}  
status = XAxiDma_SimpleTransfer(&DMAACP_instance, (UINTPTR)intermediate, (row2 * col2 + row1*col1) * sizeof(M_TYPE), XAXIDMA_DMA_TO_DEVICE); // uses axi lit  
if (status != XST_SUCCESS)  
{  
    M_TYPE intermediate[index];  
    Press 'F2' for focus  
    return 1; // returning of failure  
}  
  
// XaxiDMA_readreg(Base add. of dma)  
status = XAxiDma_ReadReg(XPAR_AXI_DMA_0_BASEADDR, 0x04) & 0x00000002;
```

SDK

The screenshot shows the Xilinx SDK IDE interface. The title bar reads "Matrix_mul.sdk - C/C++ - mat_mul/src/helloworld.c - Xilinx SDK". The menu bar includes File, Edit, Navigate, Search, Project, Run, Xilinx, Window, and Help. The toolbar contains various icons for file operations like Open, Save, and Build. The left sidebar shows the project structure with files "helloworld.c", "xaxidma.h", and "xaxidma.h". The main code editor window displays the following C code:

```
// XAxiDMA_readreg(Base add. of dma)
status = XAxiDma_ReadReg(XPAR_AXI_DMA_0_BASEADDR, 0x04) & 0x00000002;
// status=status&0x00000002;
while (status != 0x00000002)
{
    status = XAxiDma_ReadReg(XPAR_AXI_DMA_0_BASEADDR, 0x04) & 0x00000002;
}
status = XAxiDma_ReadReg(XPAR_AXI_DMA_0_BASEADDR, 0x34) & 0x00000002;
while (status != 0x00000002)
{
    status = XAxiDma_ReadReg(XPAR_AXI_DMA_0_BASEADDR, 0x34) & 0x00000002;
}
XTIME_GetTime(&DMA_ACP_end);

// Compare benchmark and hardware function output
for (i = 0; j < row1; j++)
{
    for (j = 0; j < col2; j++)
    {
        if (fabs(result_hw[i][j] - result[i][j]) > 0.01)
        {
            printf("Error at row index %d and col index %d\n", i, j);
            printf("Hardware output %f\n", result_hw[i][j]);
            printf("software output %f\n", result[i][j]);
            return 1;
        }
    }
}
printf("No Error!!");

// PS time
printf("for Iteration %d\n", a);
printf("-----\n");
float time_processor_PS_0 = 0;
```

Performance gain

- Matrix multiplication of order(8,8)
- By parallelization (via pragma)

	Processing system	Programmable logic
Time (in us)	26.12	6.95

Performance gain

- Eg. Matrix multiplication of order(8,8)
- By parallelization (via pragma)

	Processing system	Programmable logic
Time (in us)	26.12	6.95

Throughput improvement 3.7x

Performance gain

- Eg. Least Square Deep Neural Network Estimation.

	Processing system	Programmable logic
Time (in us)	87684.39	15967.95

Performance gain

- Eg. Least Square Deep Neural Network Estimation.

	Processing system	Programmable logic
Time (in us)	87684.39	15967.95

Throughput improvement 5.5x

Performance comparison

Features	Processing system (FPGA)	x86 processor
frequency	300 MHz	3-4 GHz
Power	1-4 W	50-110 W

Performance comparison

Features	Processing system	x86 processor
frequency	600 MHz	3-4 GHz
Power	1-4 W	50-110 W

- Edges device cannot be bulky and power hungry
- Limited compute capabilities

Why to use FPGA?

- To reduce load from CPU (by offloading application to smartNIC-FPGA)
- Edge device can perform computation.

Feedback

- Quiz

Quiz

Q1) Who is responsible for last signal at a given instance?

- a) Sender
- B) Receiver
- C) Both

Q2) Can we find the address of the data in stream based protocol?

Yes/No

Q3) Is FPGA programmable?

Yes/No

Quiz

Q1) Who is responsible for last signal?

- a) **Sender**
- B) Receiver
- C) Both

Q2) Can we find the address of the data in stream based protocol?

Yes/No

Q3) Is FPGA programmable?

Yes/No

Quiz

Q1) Who is responsible for last signal at a given instance?

- a) **Sender**
- B) Receiver
- C) Both

Q2) Can we find the address of the data in stream based protocol?

No

Q3) Is FPGA programmable?

Yes/No

Quiz

Q1) Who is responsible for last signal at a given instance?

- a) **Sender**
- B) Receiver
- C) Both

Q2) Can we find the address of the data in stream based protocol?

No

Q3) Is FPGA programmable?

Yes

Quiz

Q4) Whose frequency is greater **x86 processor or FPGA?**

Q5) Why DMA is required between PS and PL?

Q6) To which data-structure stream characteristics matches?

- a) Stack
- b) Queue
- c) Array

Quiz

Q4) Whose frequency is greater **x86 processor or FPGA?**

x86 processor

Q5) Why DMA is required between PS and PL?

Q6) To which data-structure stream characteristics matches?

- a) Stack
- b) Queue
- c) Array

Quiz

Q4) Whose frequency is greater **x86 processor or FPGA?**

x86 processor

Q5) Why DMA is required between PS and PL?

DMA translate from memory mapped to stream based and vice-versa

Q6) To which data-structure stream characteristics matches?

- a) Stack
- b) Queue
- c) Array

Quiz

Q4) Whose frequency is greater **x86 processor or FPGA?**

x86 processor

Q5) Why DMA is required between PS and PL?

DMA translate from memory mapped to stream based and vice-versa

Q6) To which data-structure stream characteristics matches?

- a) ~~Stack~~
- b) Queue i.e FIFO**
- c) ~~Array~~