Getting started with FPGA

Rinku Shah[1], Assistant Professor, IIIT Delhi
Neeraj Kumar Yadav[1], Research Scholar, IIIT Delhi
Keshav Gambhir[2], Microsoft India
Harish S A[3], Research Scholar (PMRF), IIT Hyderabad

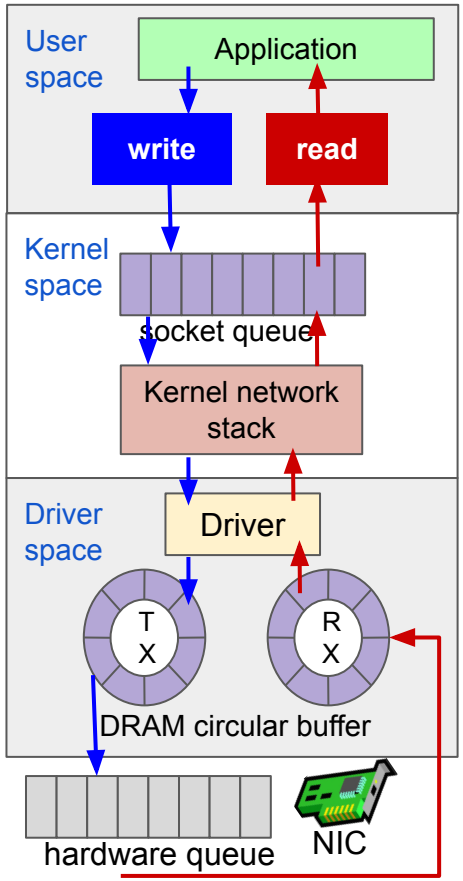
[1] IIIT Delhi
India

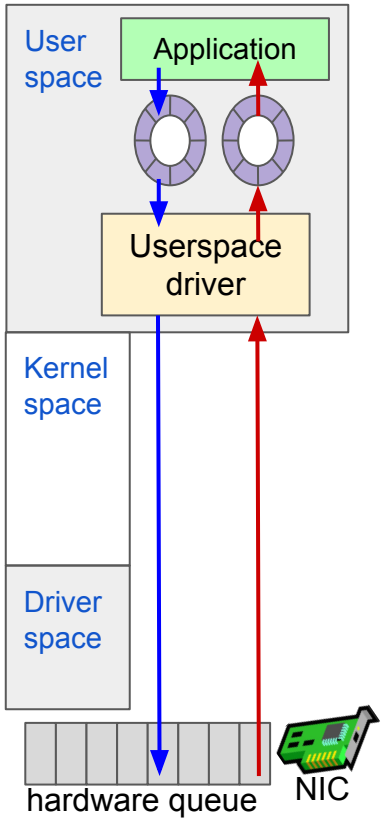
[2] Microsoft
India

[3] IIT Hyderabad
India

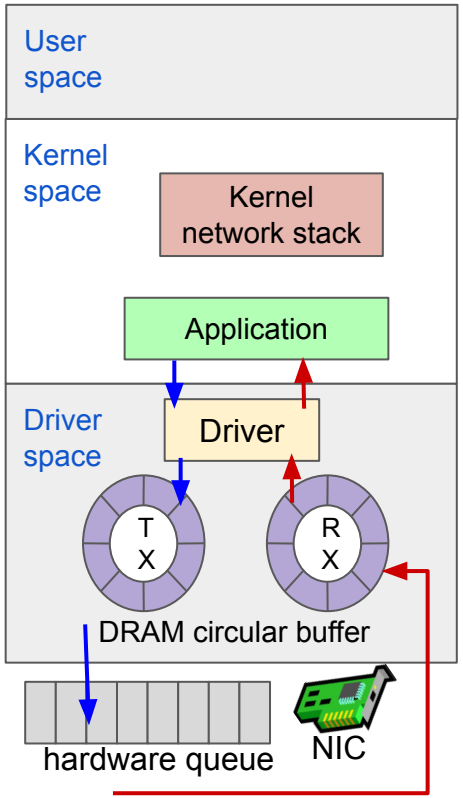
The material for this presentation is taken from various books, courses and Xilinx/ARM XUP resources.
The presenter does not claim ownership of the material presented in this workshop.

Evolution of network packet processors

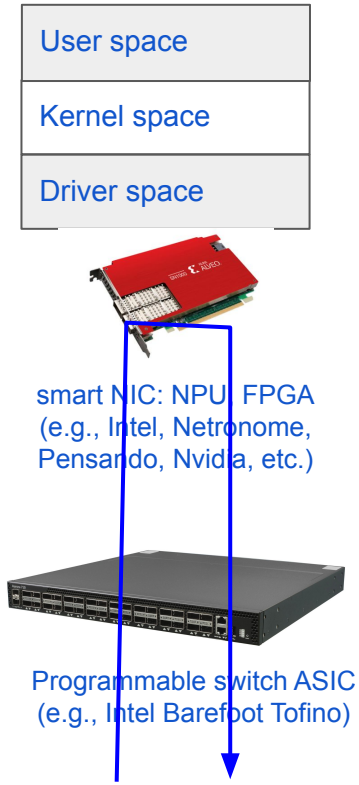




Kernel bypass (e.g. netmap, DPDK)

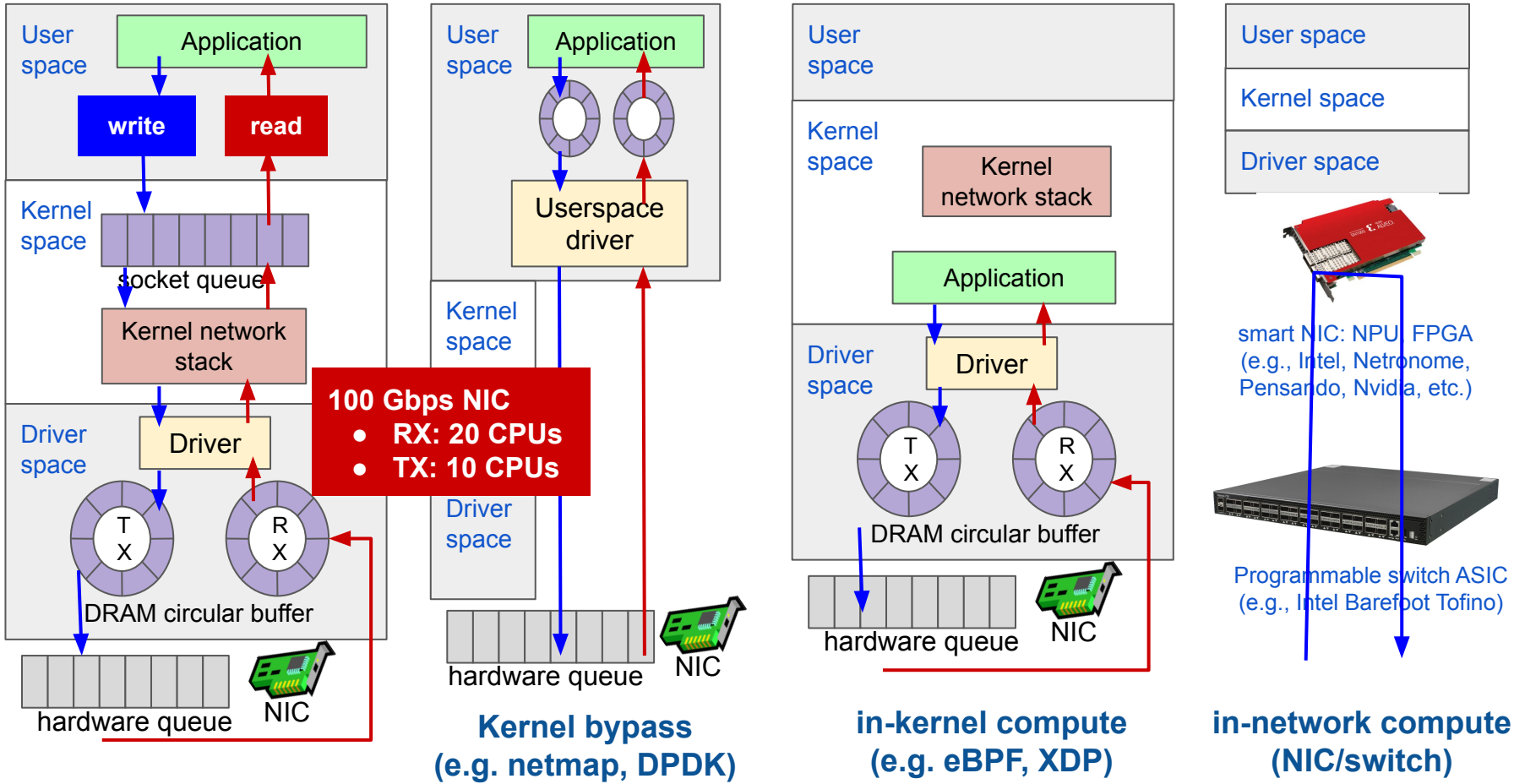


in-kernel compute (e.g. eBPF, XDP)

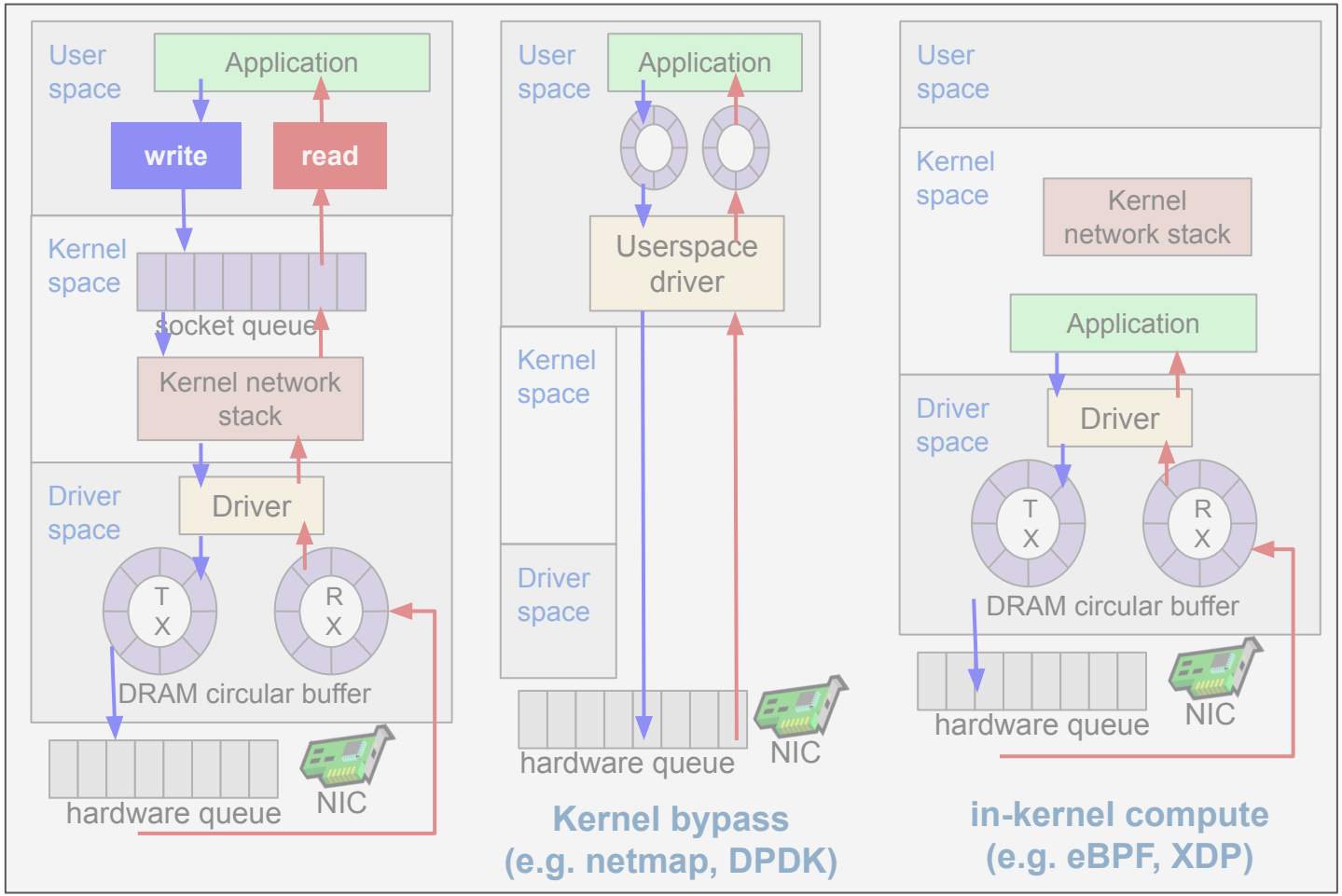


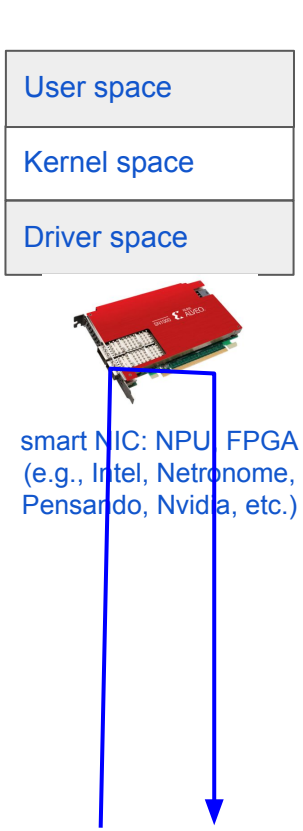
in-network compute (NIC/switch)

Evolution of network packet processors

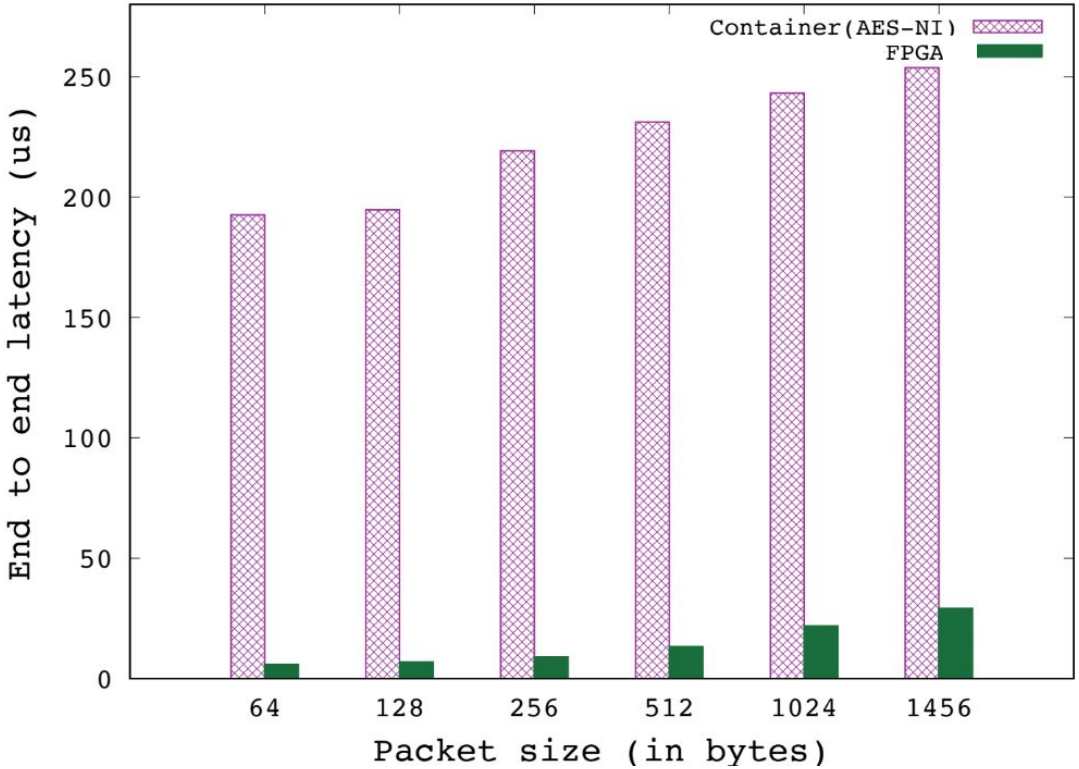


Evolution of network packet processors



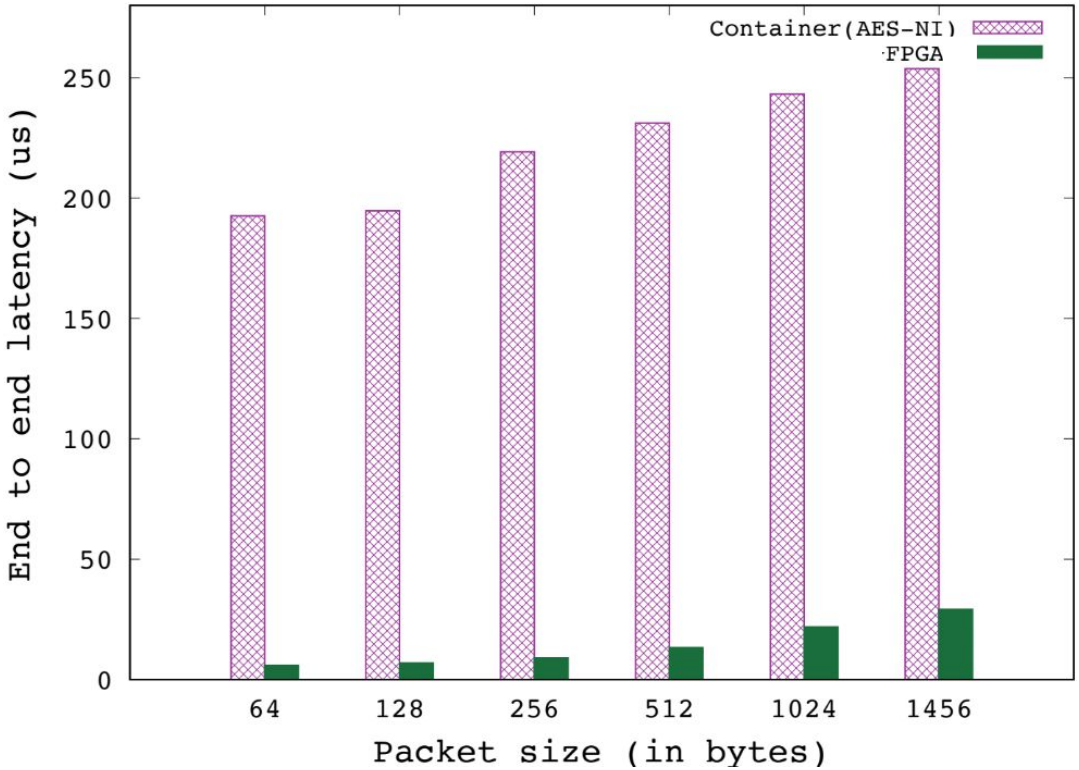


Offloading application to FPGA NIC



AES-GCM: Cryptography algorithm used in TLS.

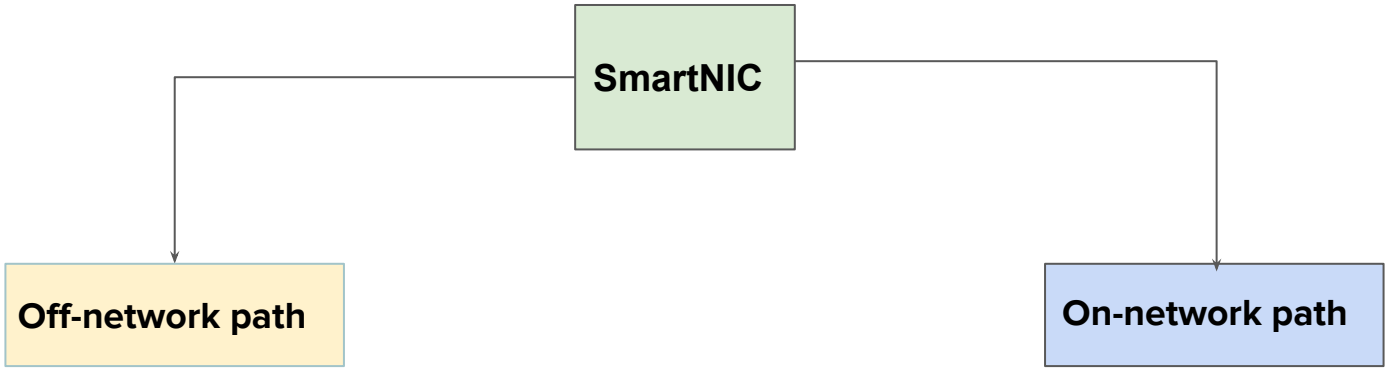
Offloading application to FPGA NIC



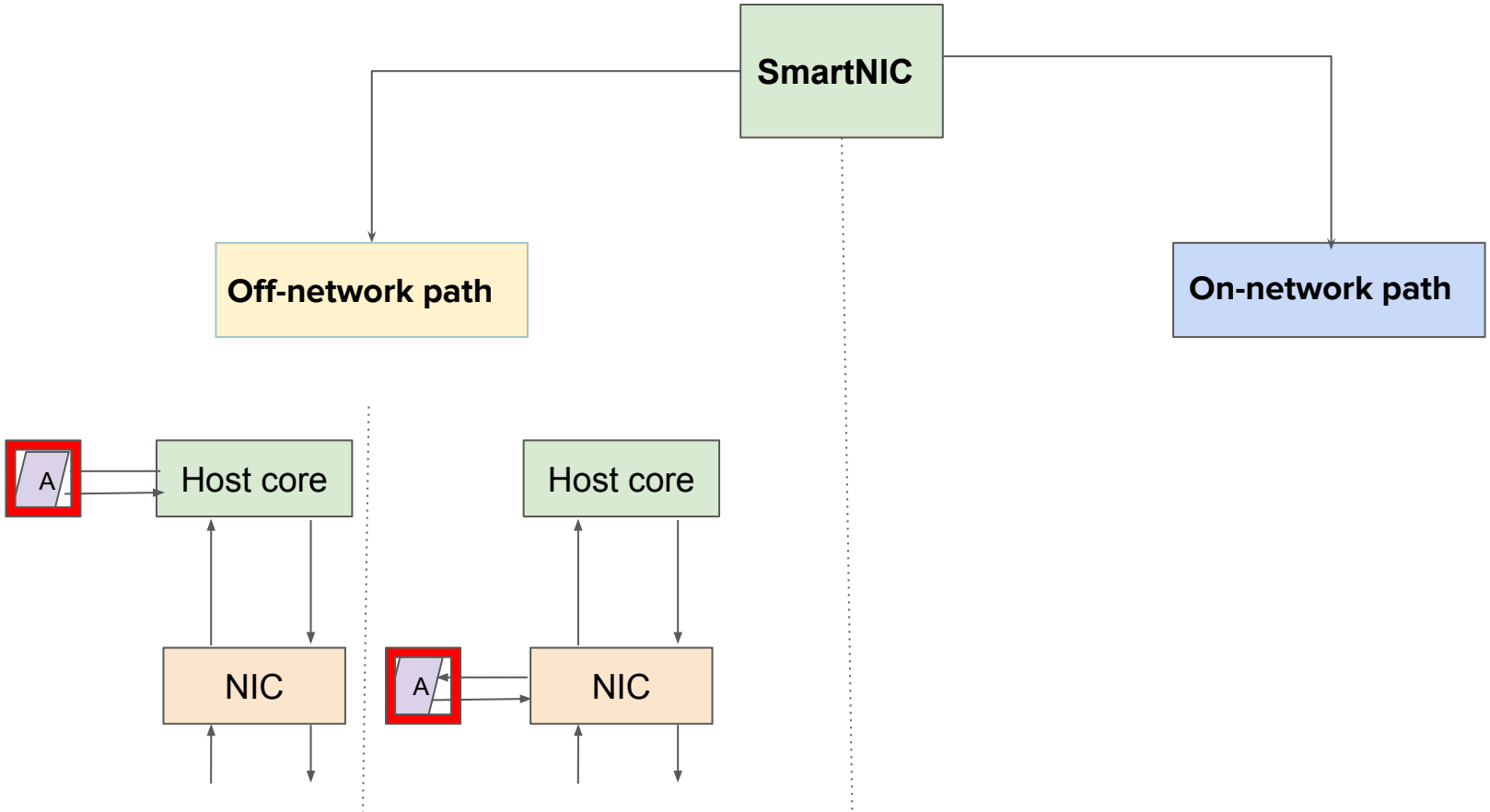
AES-GCM: Cryptography algorithm used in TLS.

Latency reduction 88.5% to 97.3%

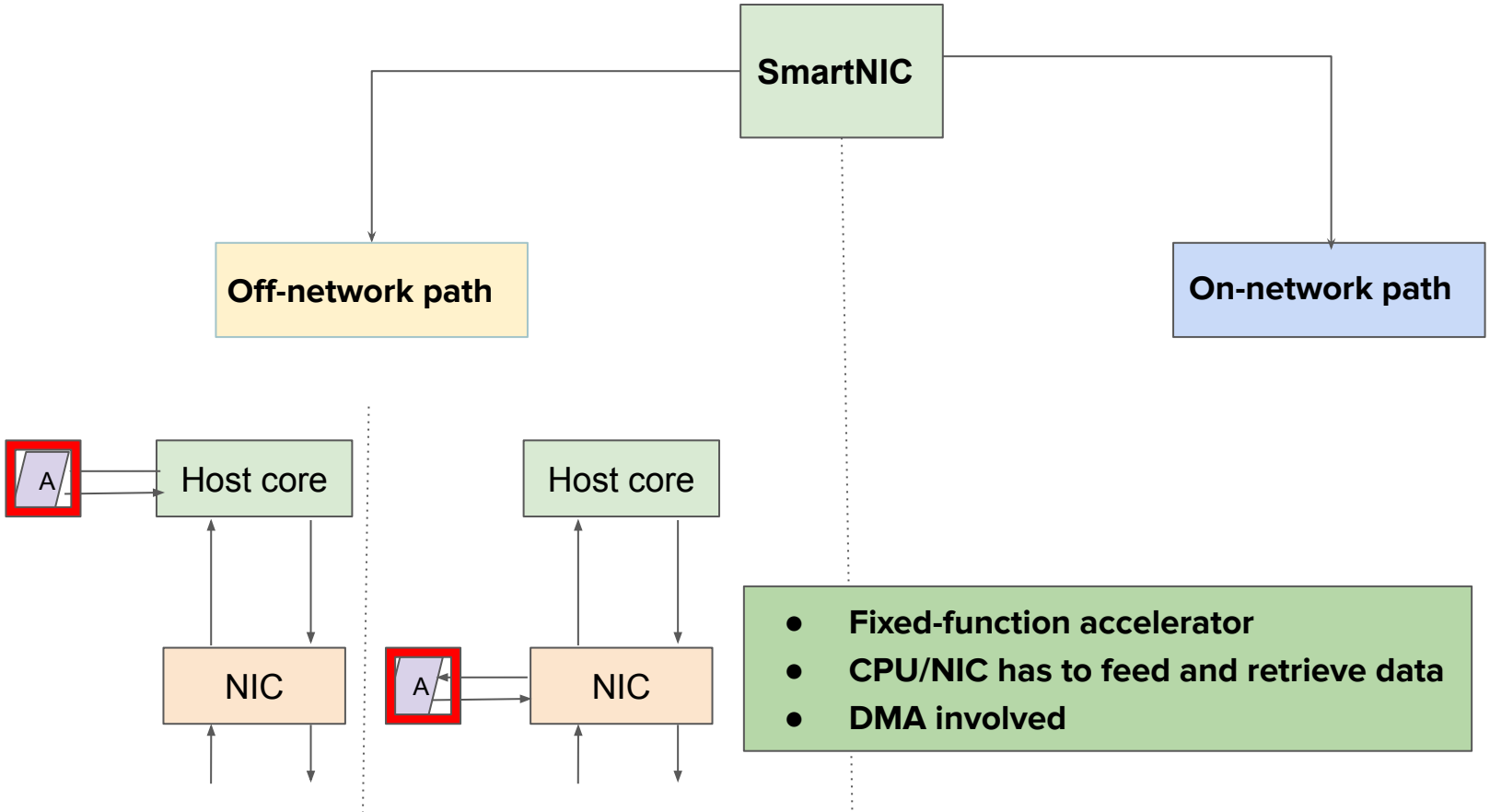
SmartNIC classification



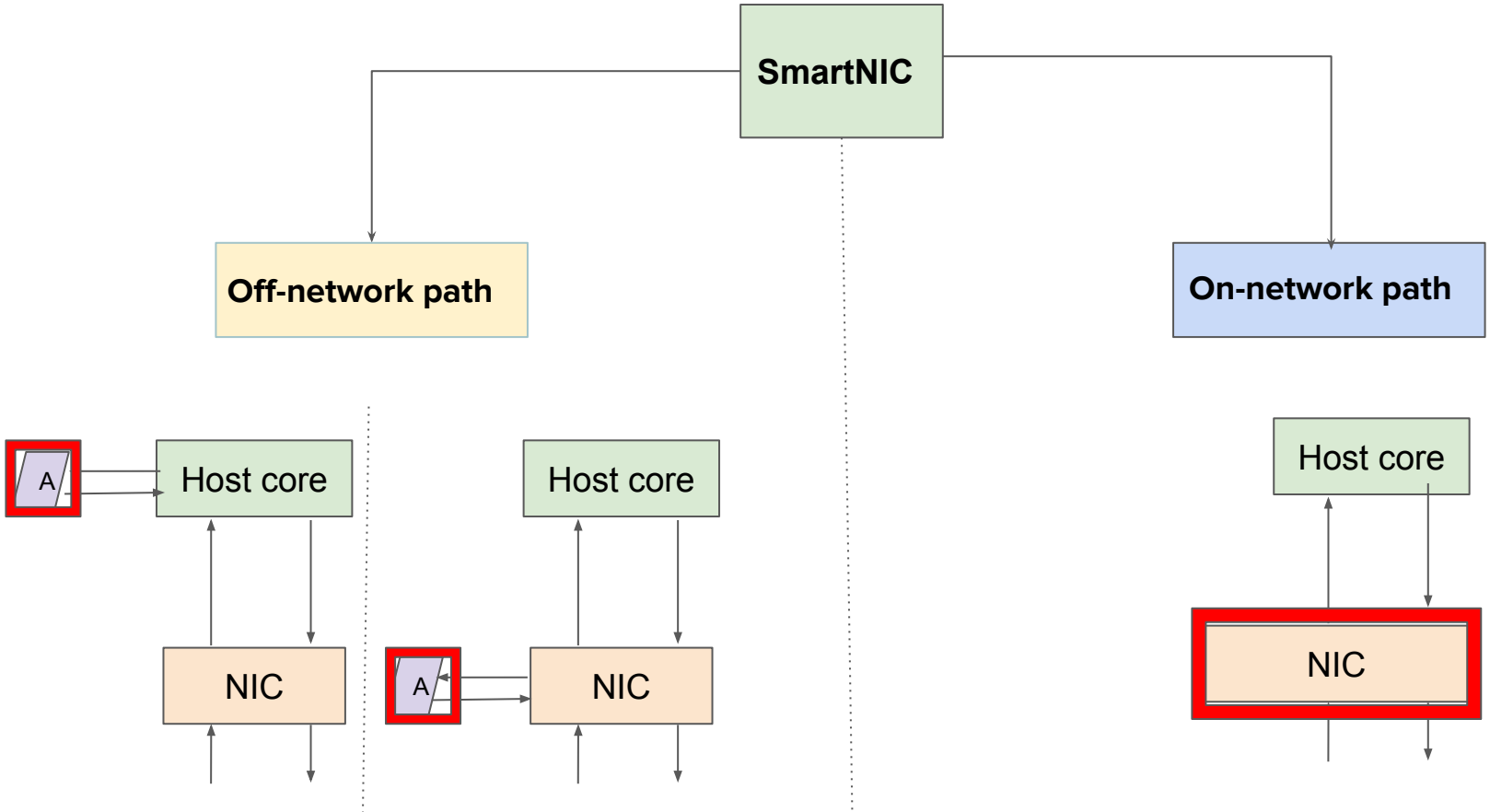
SmartNIC classification



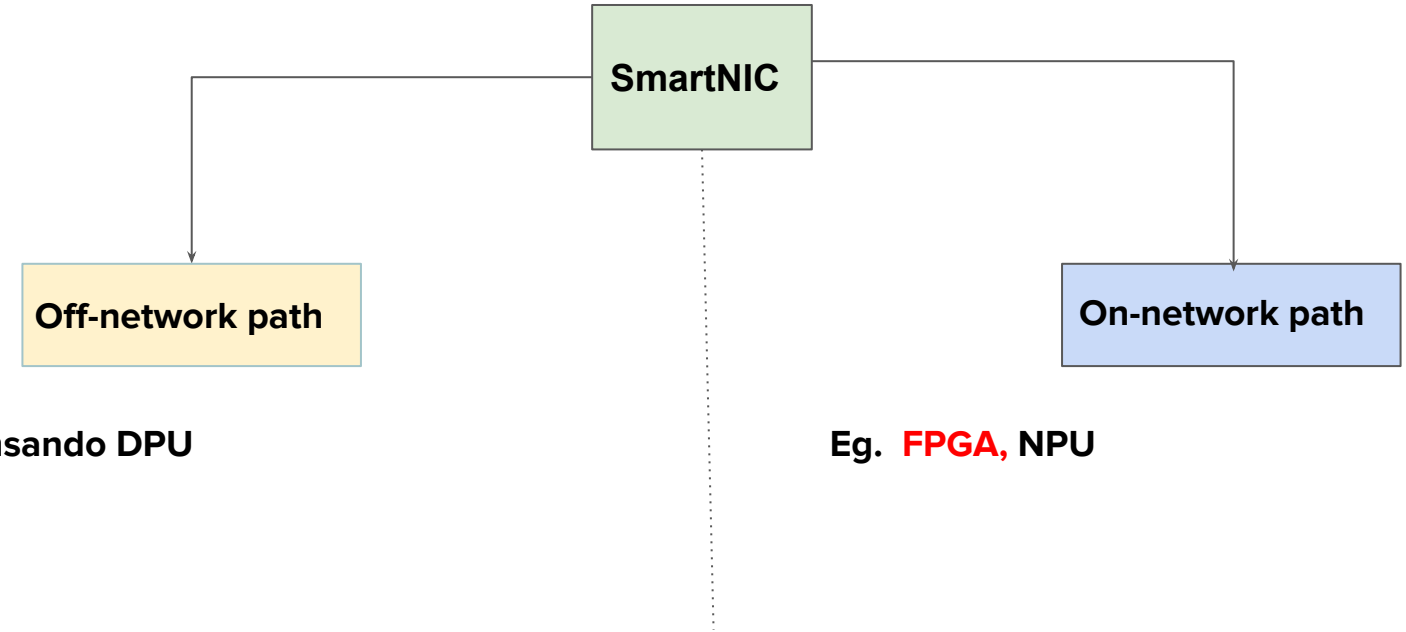
SmartNIC classification



SmartNIC classification



SmartNIC classification

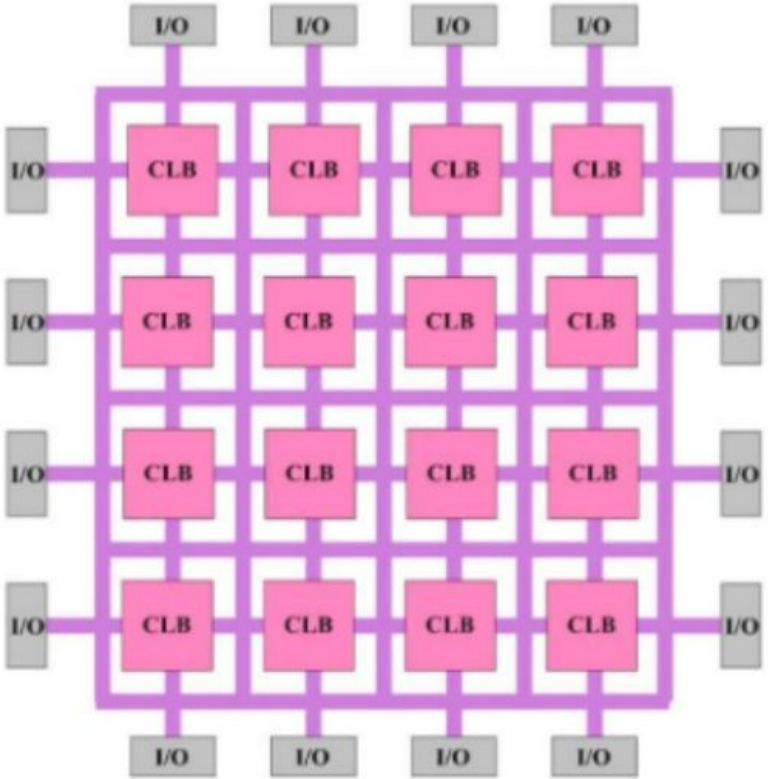


NPU-Network processing unit
DPU-Data processing unit
FPGA- Field programmable gate array

Outline

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

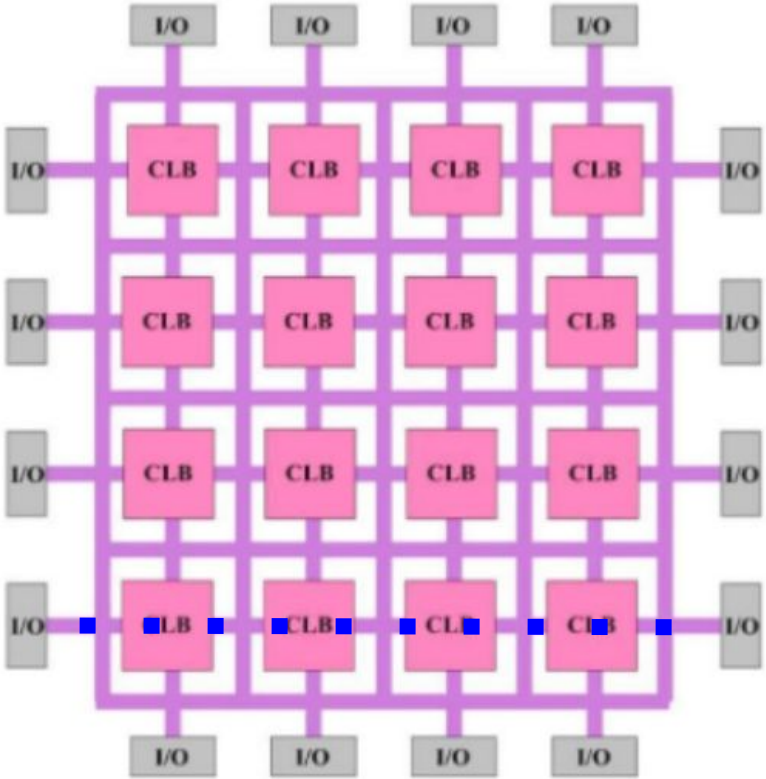
What is Field programmable gate array?



- ❏ 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

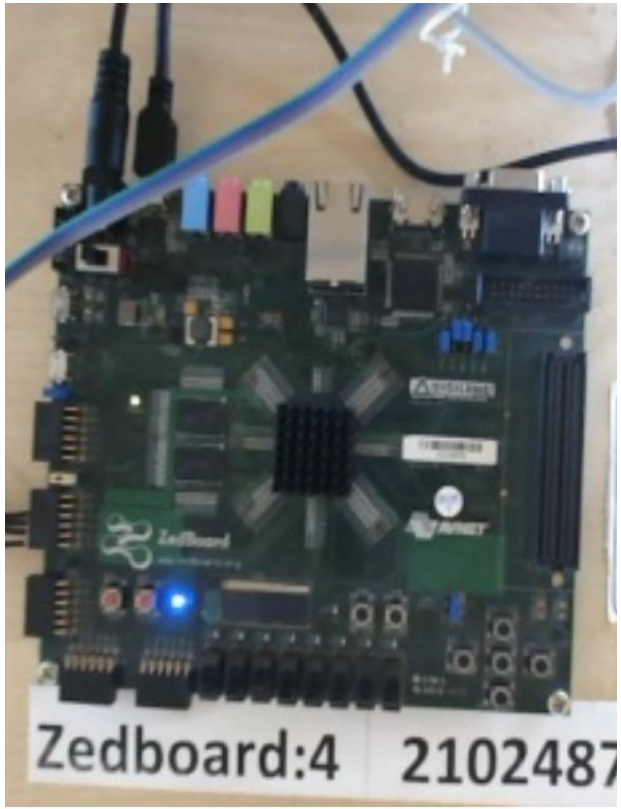
[1] CoQube :https://coqube.com/fpga-design-flow/

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

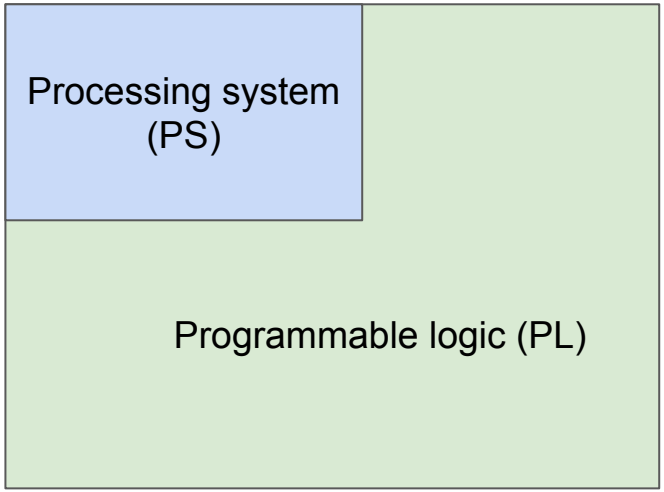
- ZCU 106

- 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

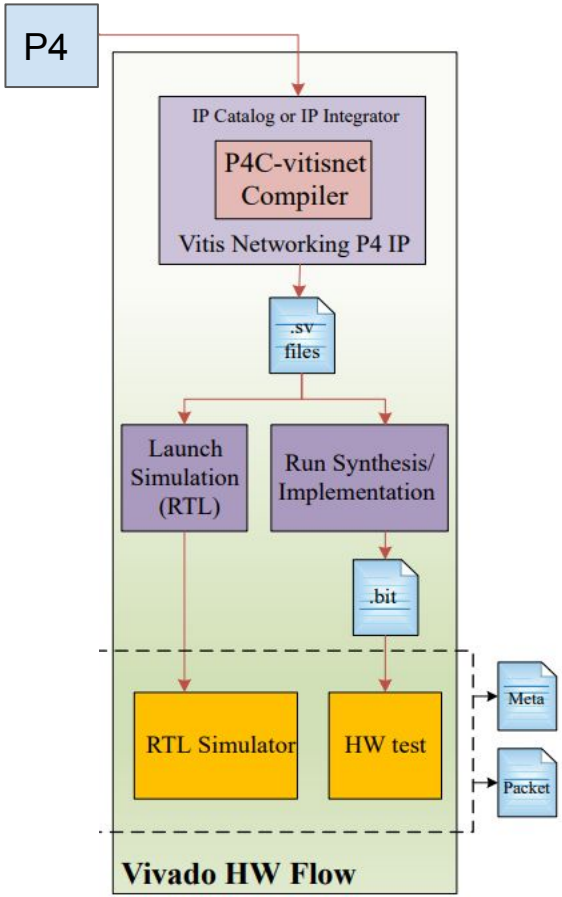
- VHDL
- Verilog

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, ...

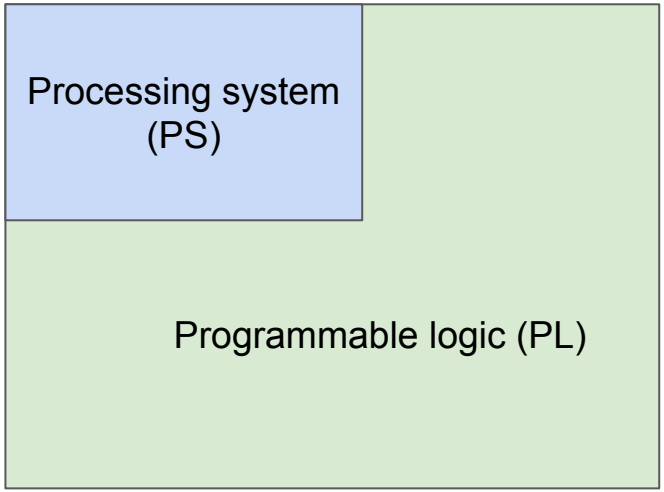
P4 to FPGA program

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

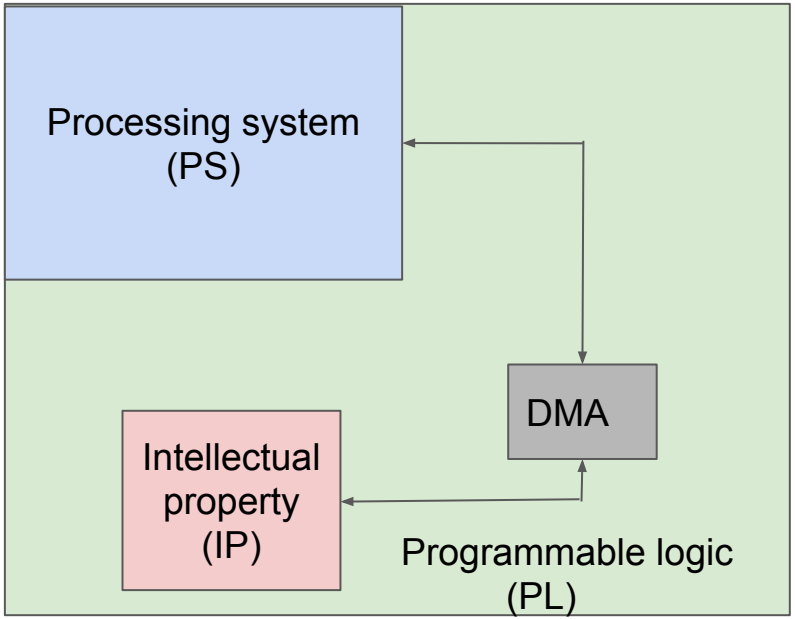
- VHDL
- Verilog

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

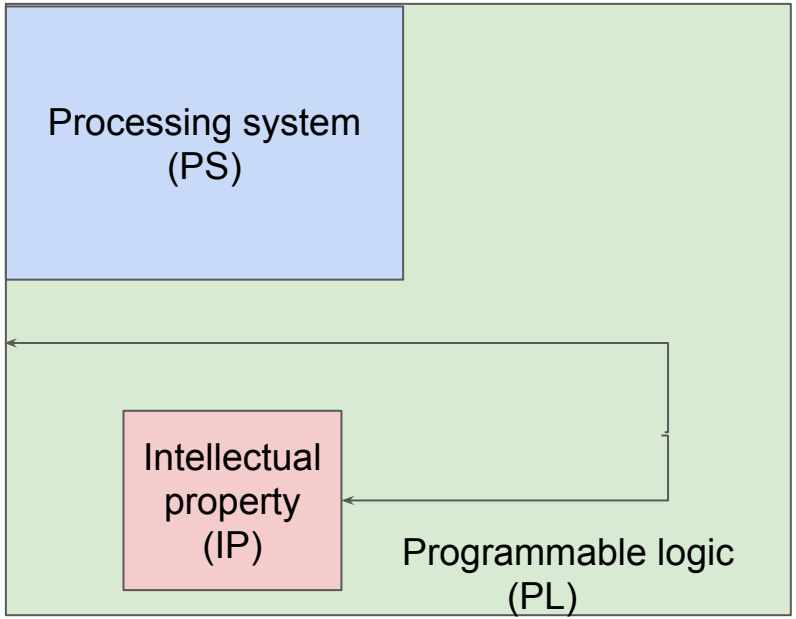
FPGA end-to-end working



- **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

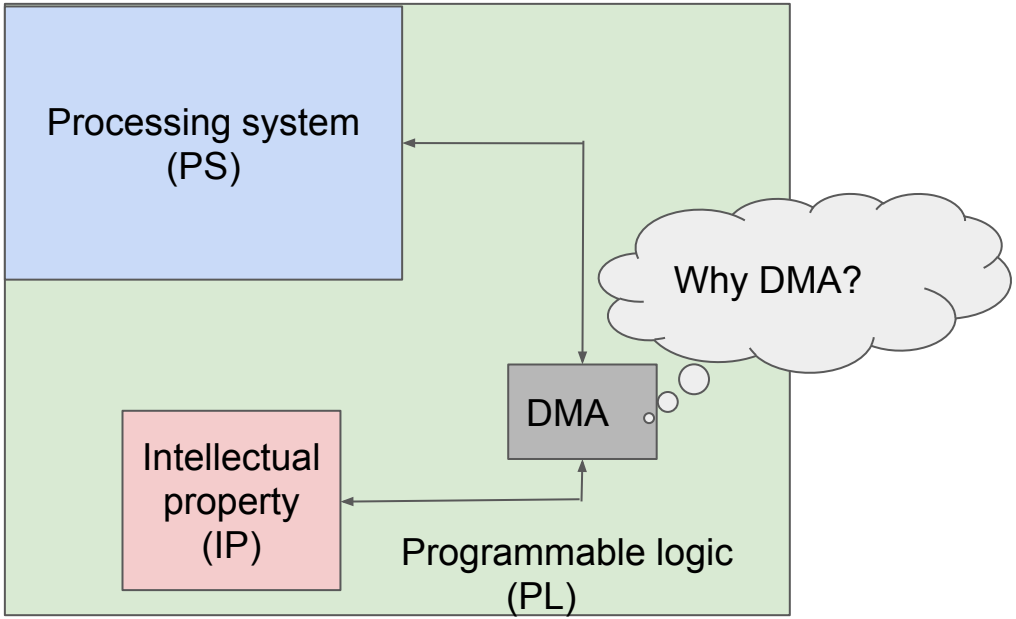
FPGA end-to-end working



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

*DMA- Direct memory access

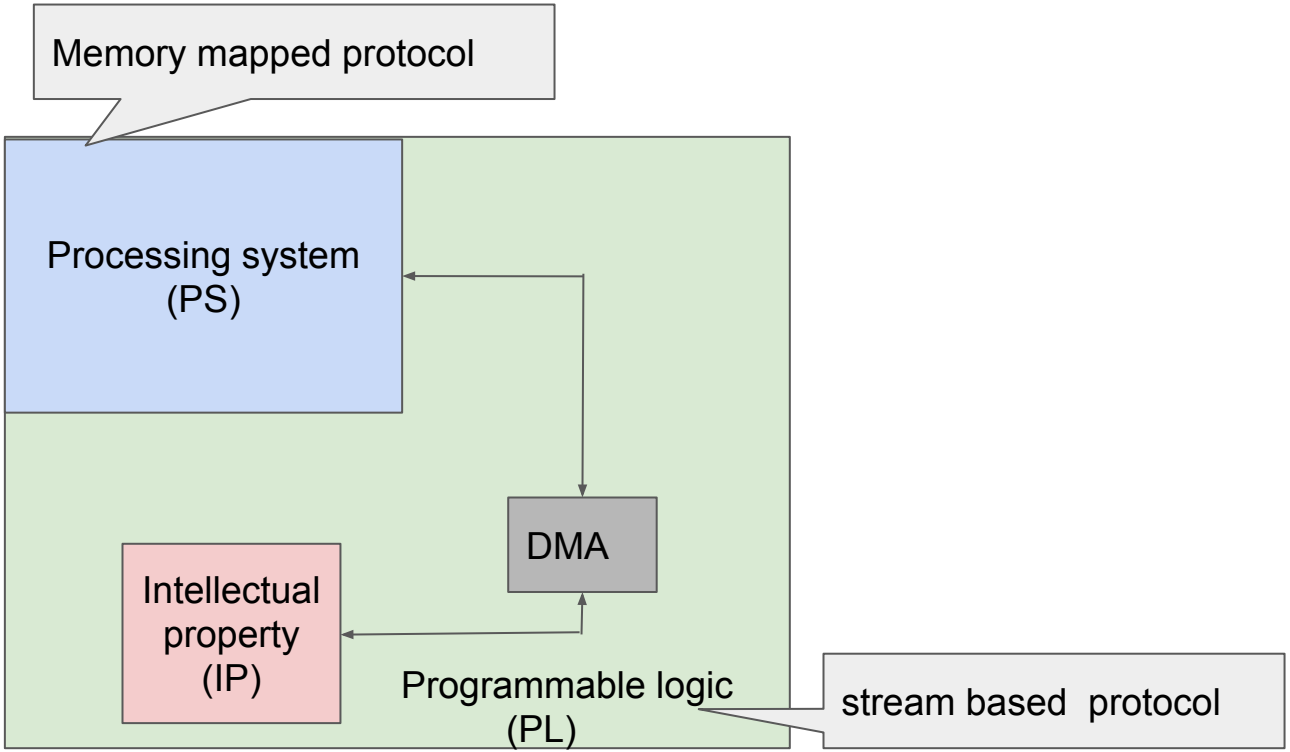
FPGA end-to-end working



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

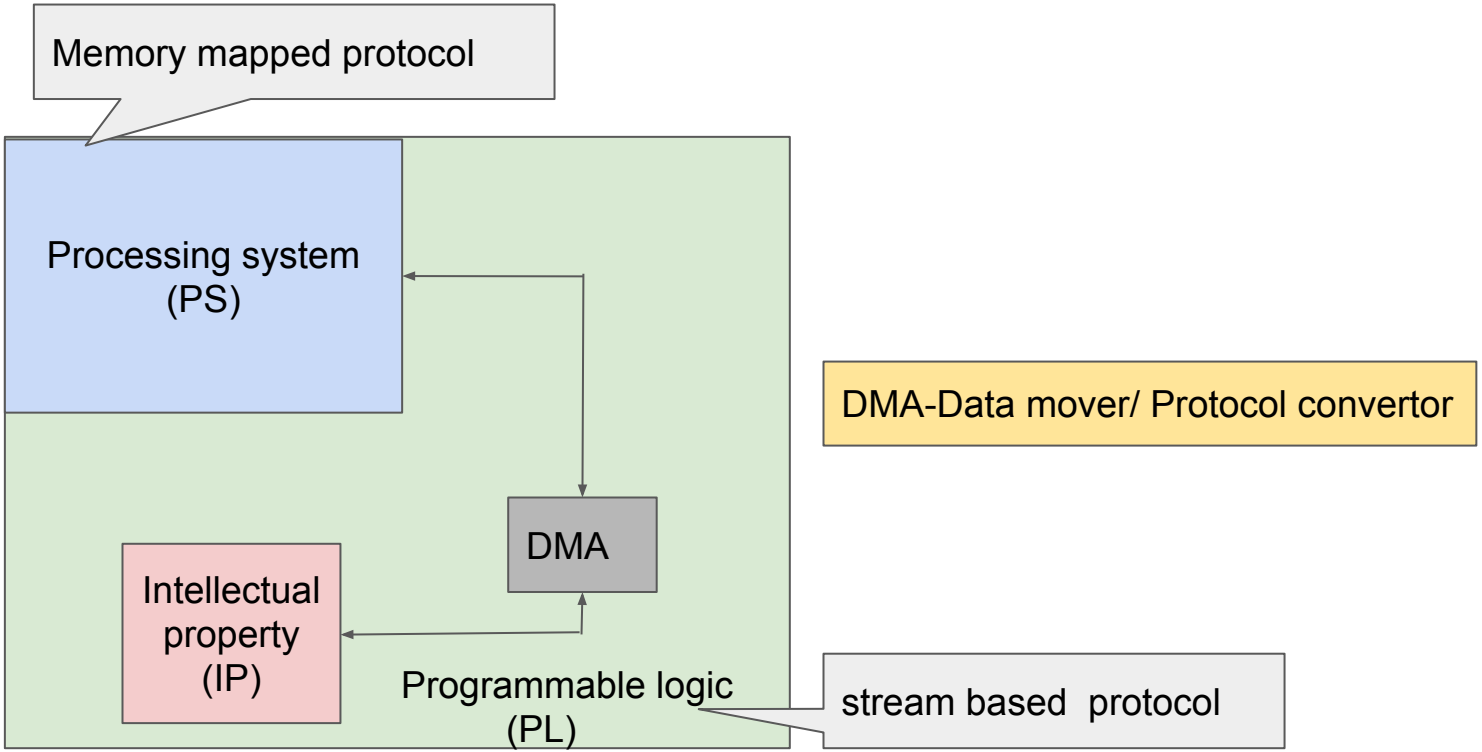
*DMA- Direct memory access

FPGA end-to-end working



*DMA- Direct memory access

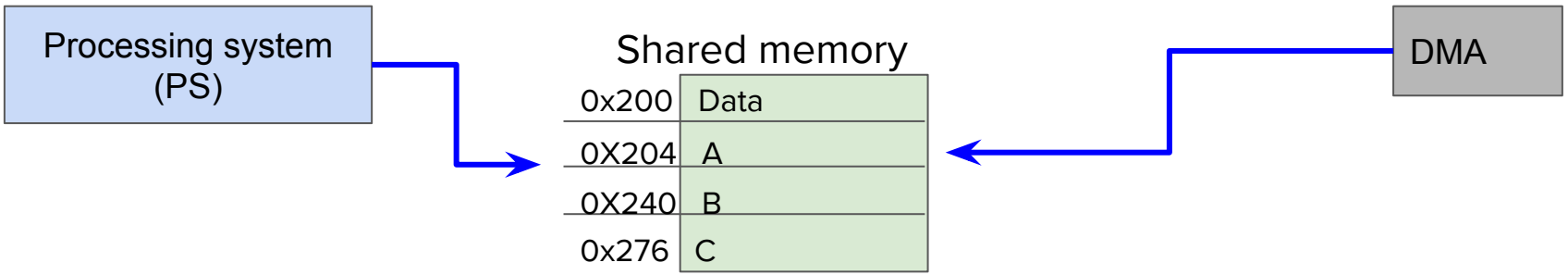
FPGA end-to-end working



*DMA- Direct memory access

Memory mapped (mm) protocol

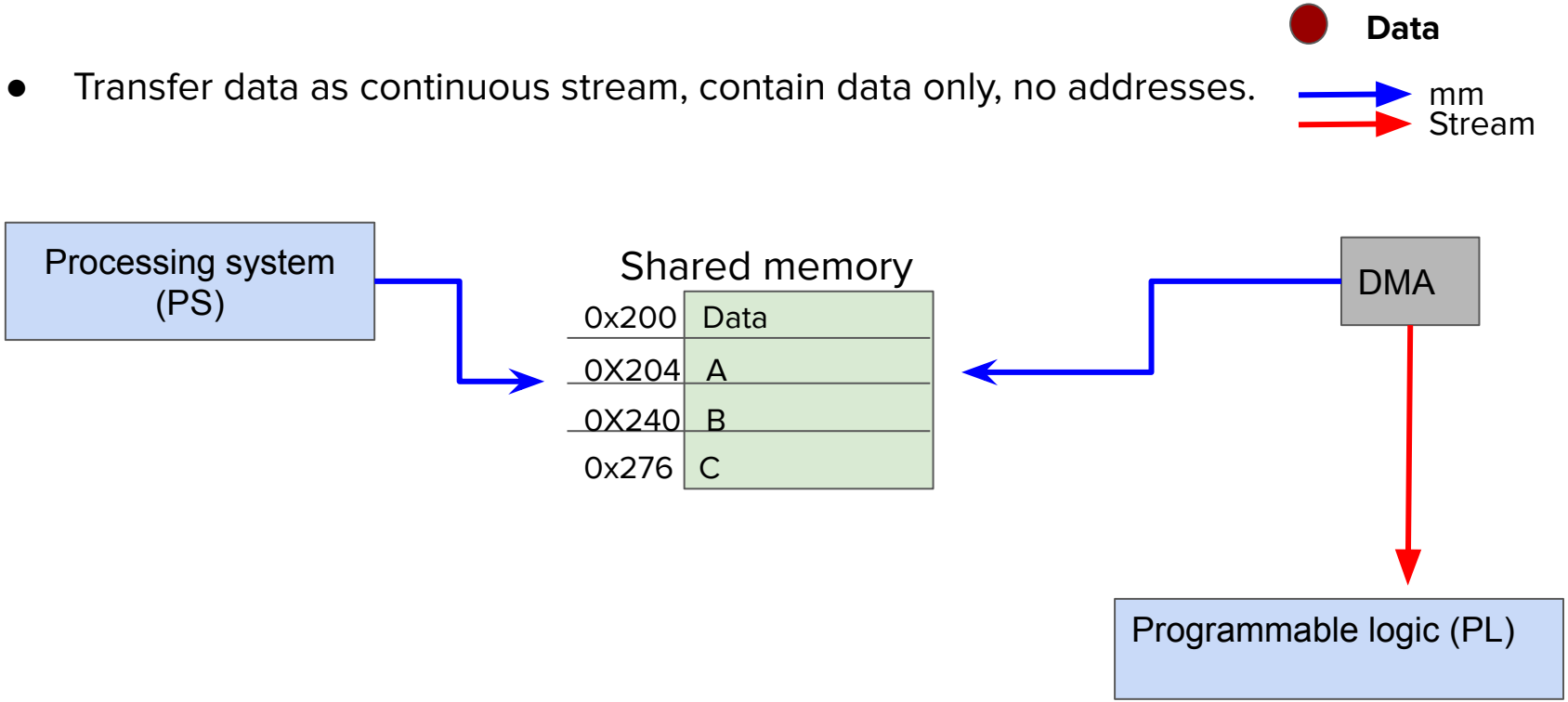
- Shared memory access between processing system and DMA
- Data and its corresponding addresses.



*DMA- Direct memory access

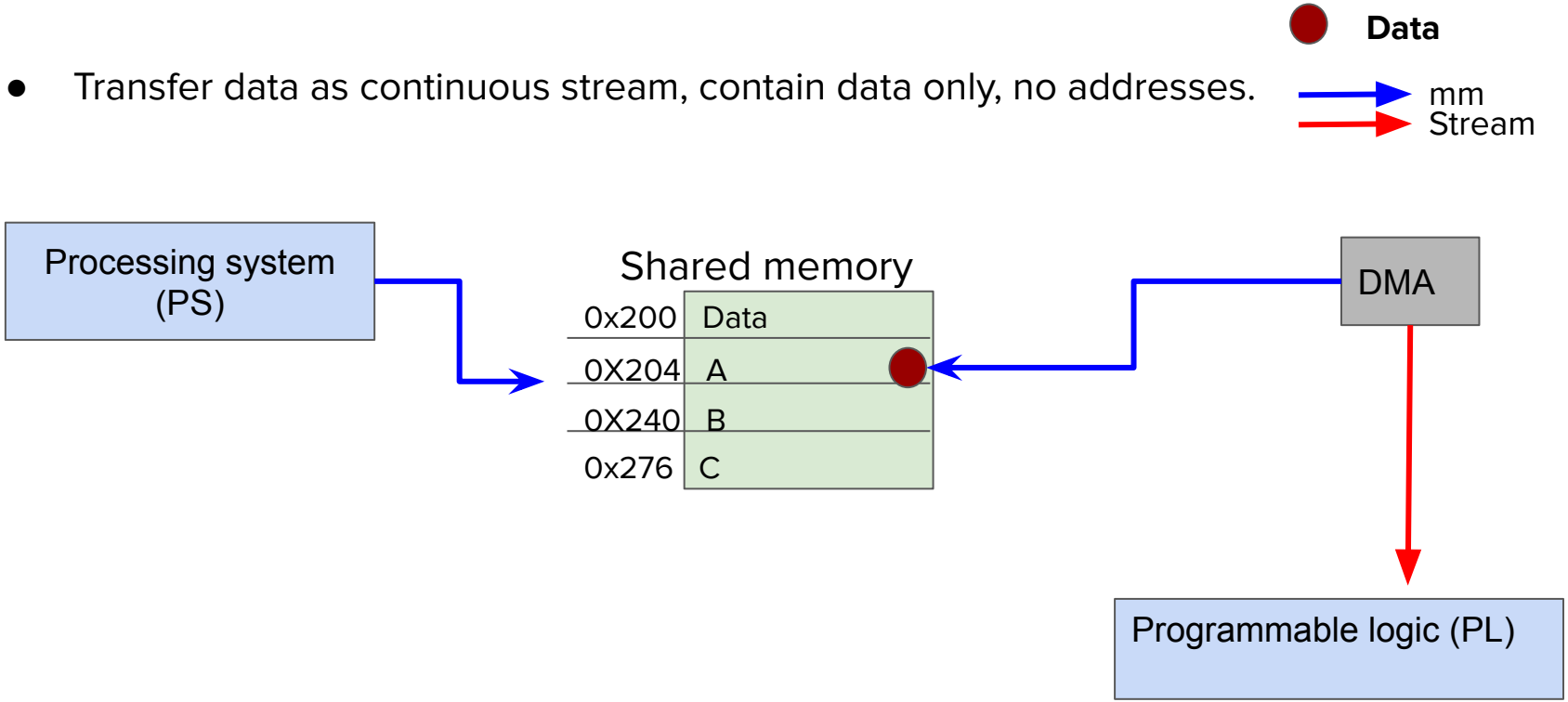
Stream based protocol

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



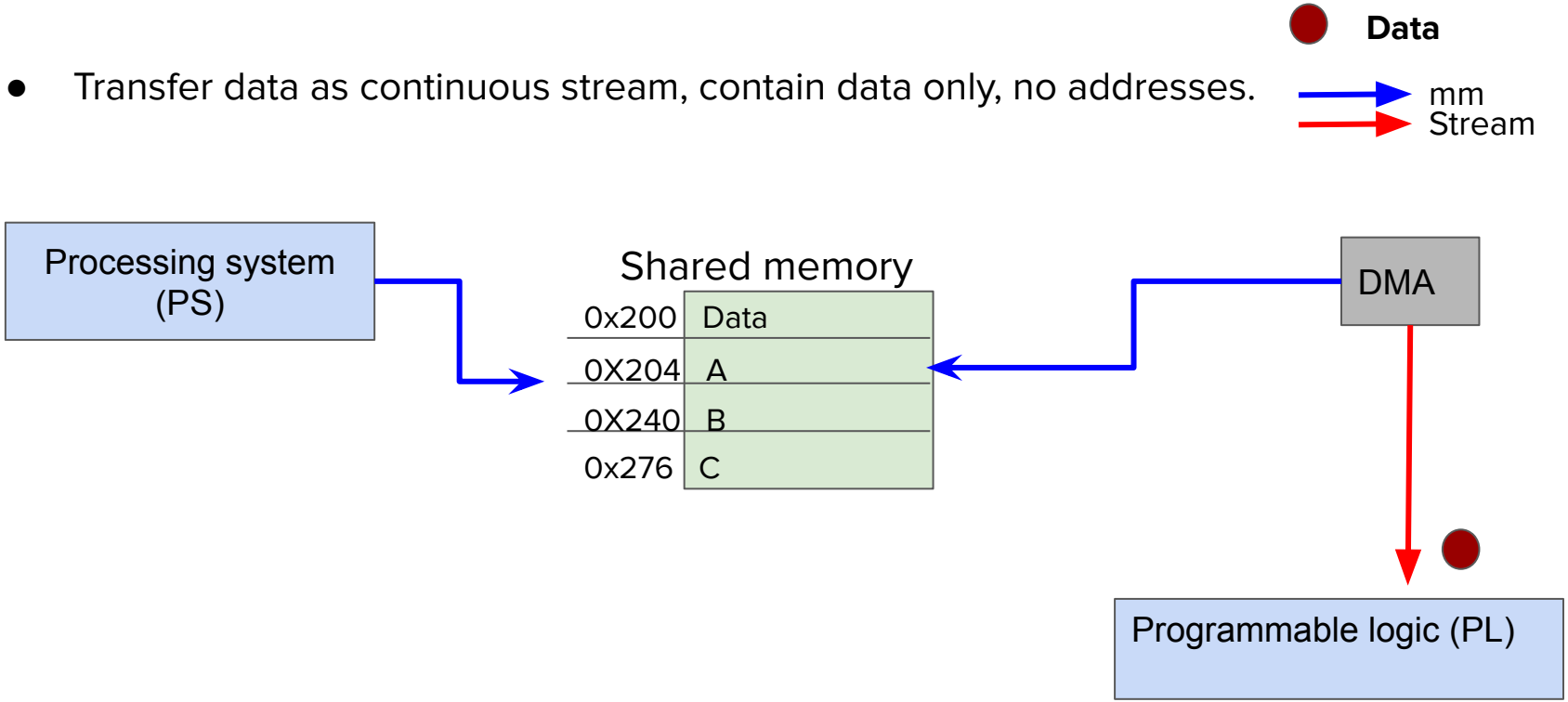
Stream based protocol

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



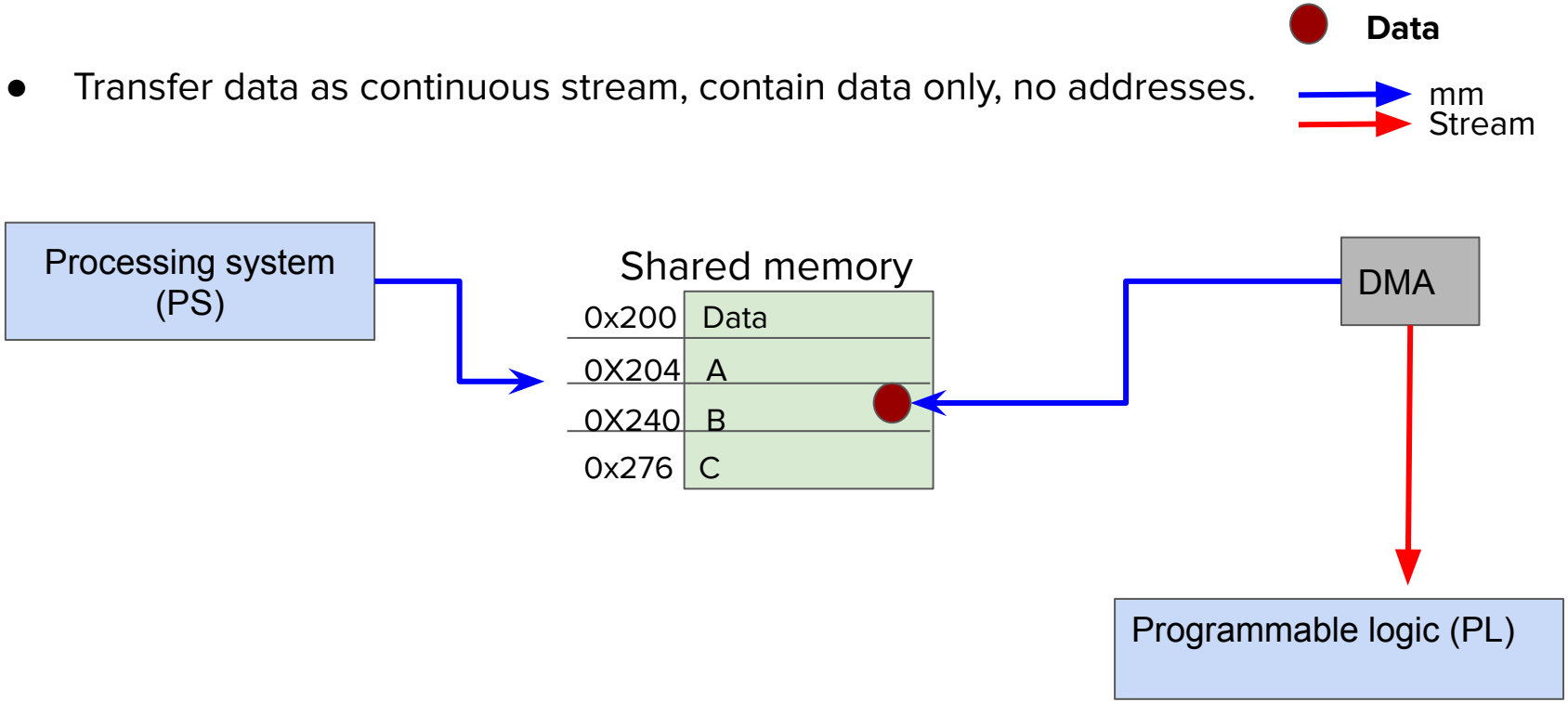
Stream based protocol

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



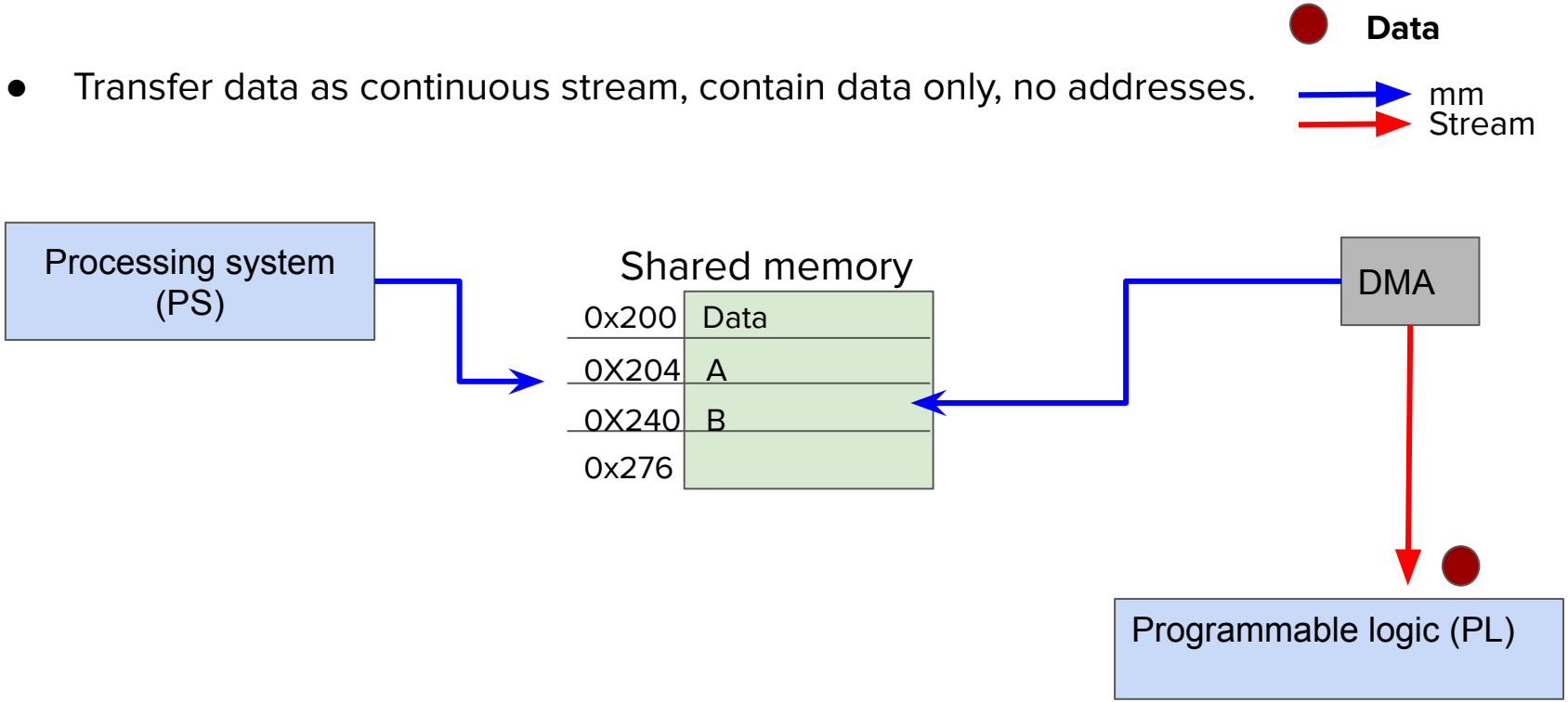
Stream based protocol

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



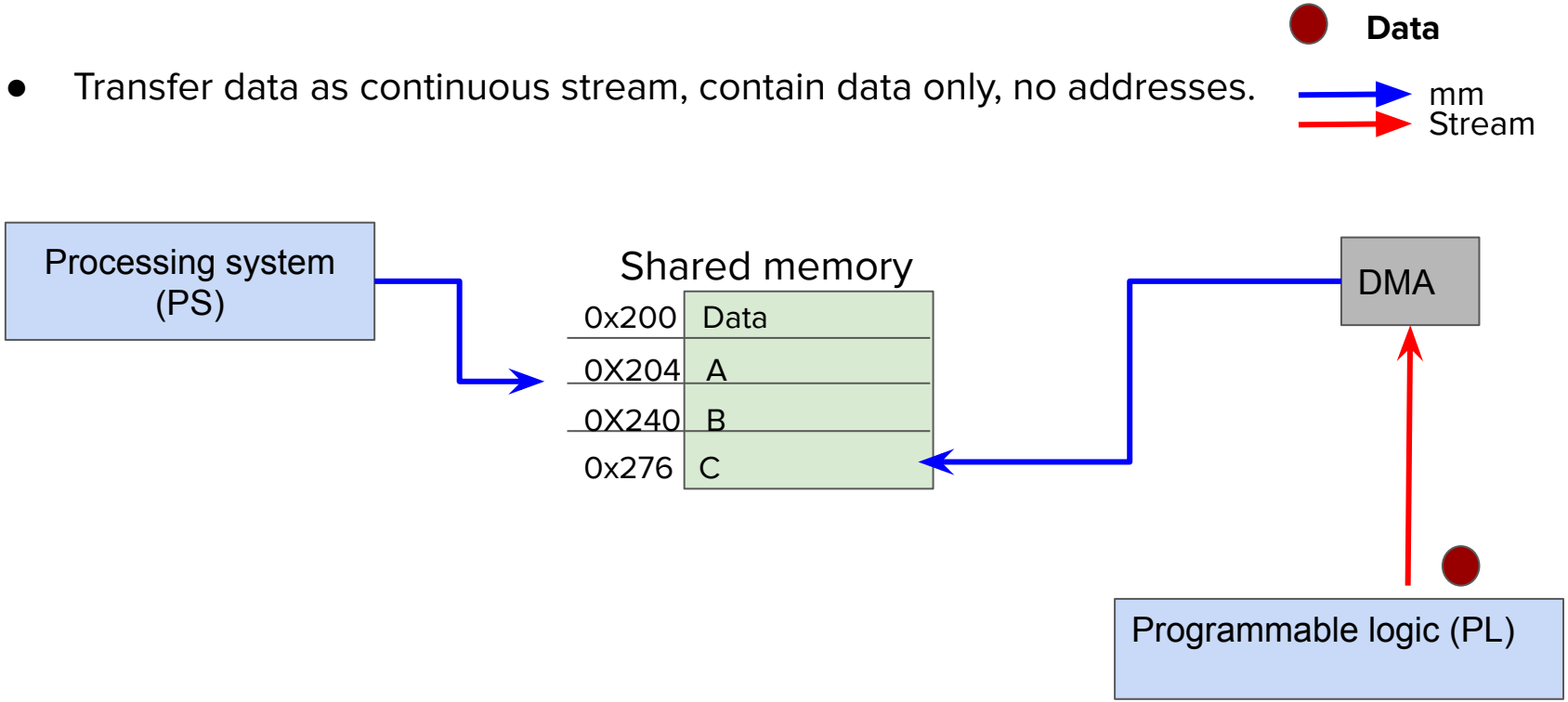
Stream based protocol

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



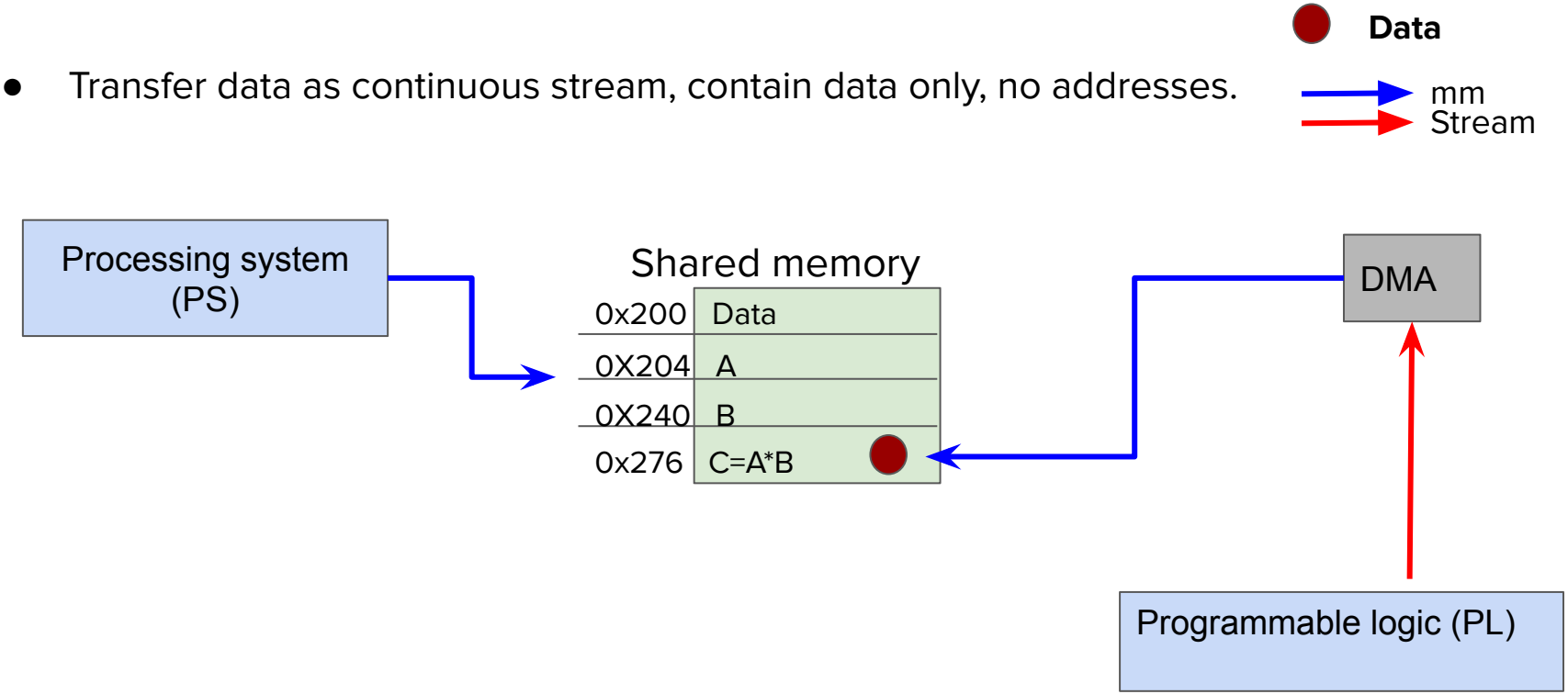
Stream based protocol

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



Stream based protocol

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



What is memory mapped and stream based protocol?

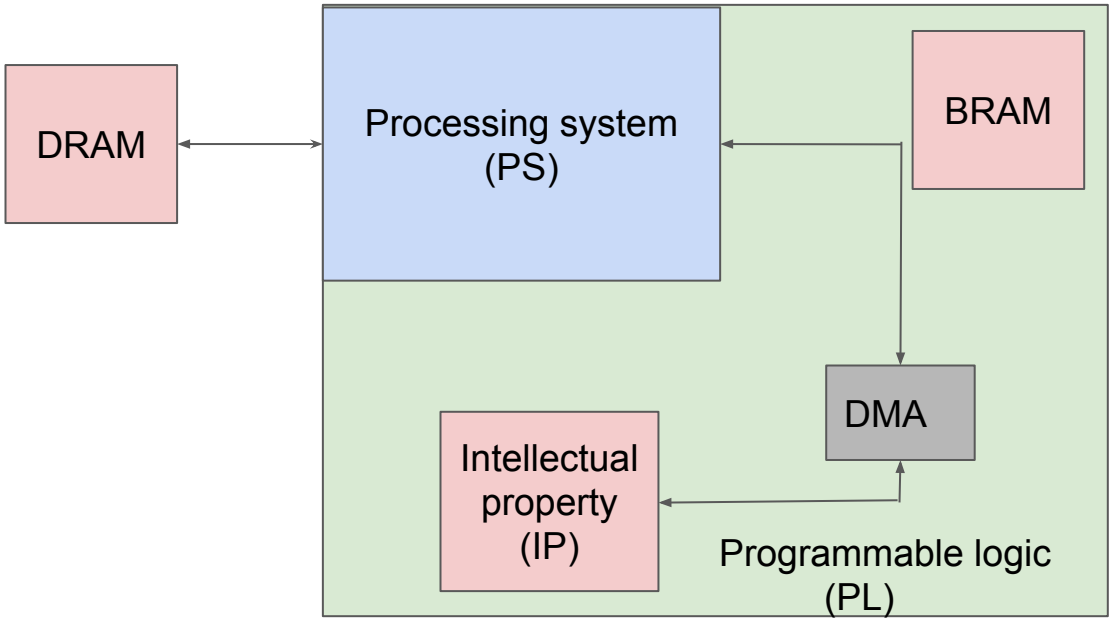
Memory mapped

- Data and its corresponding addresses.
- Shared memory access

Stream based

- Transfer data as continuous stream
- Contains data only.

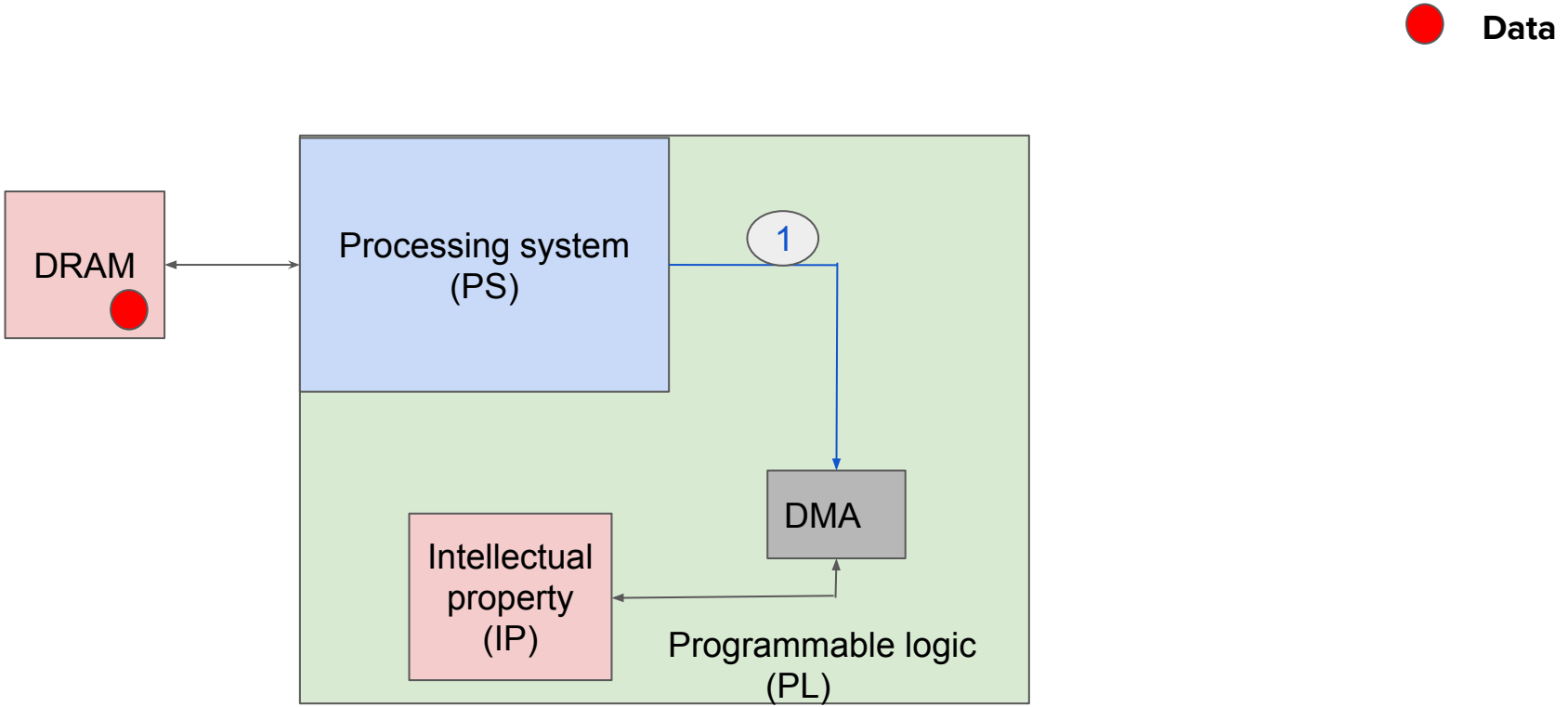
FPGA: Memories



- **BRAM has less latency than DRAM**

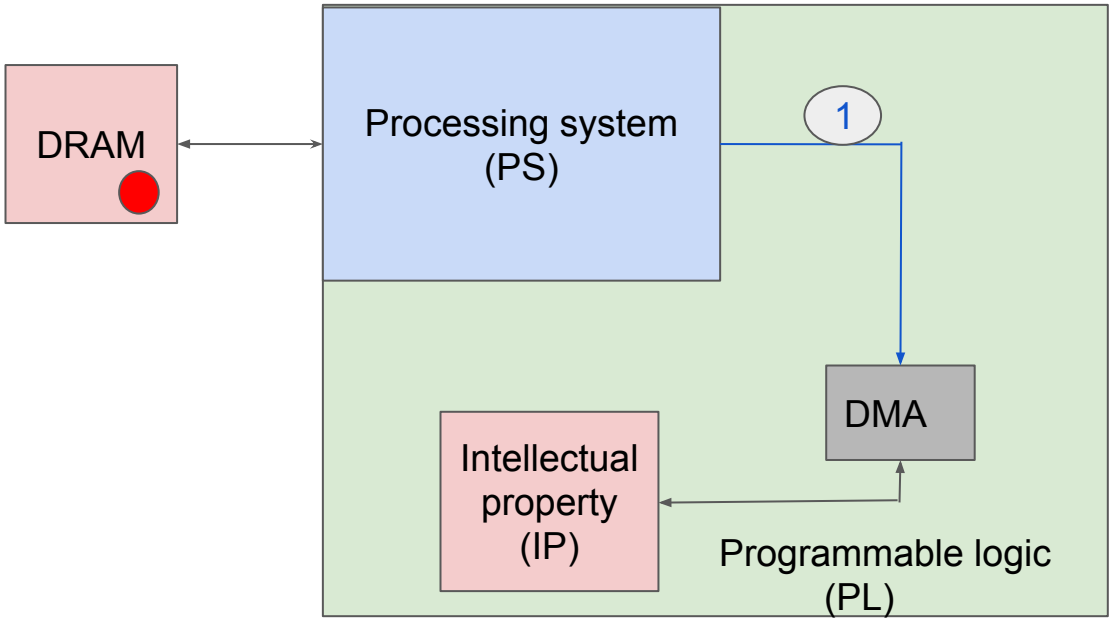
*DMA- Direct memory access

FPGA end-to-end flow of data



*DMA- Direct memory access

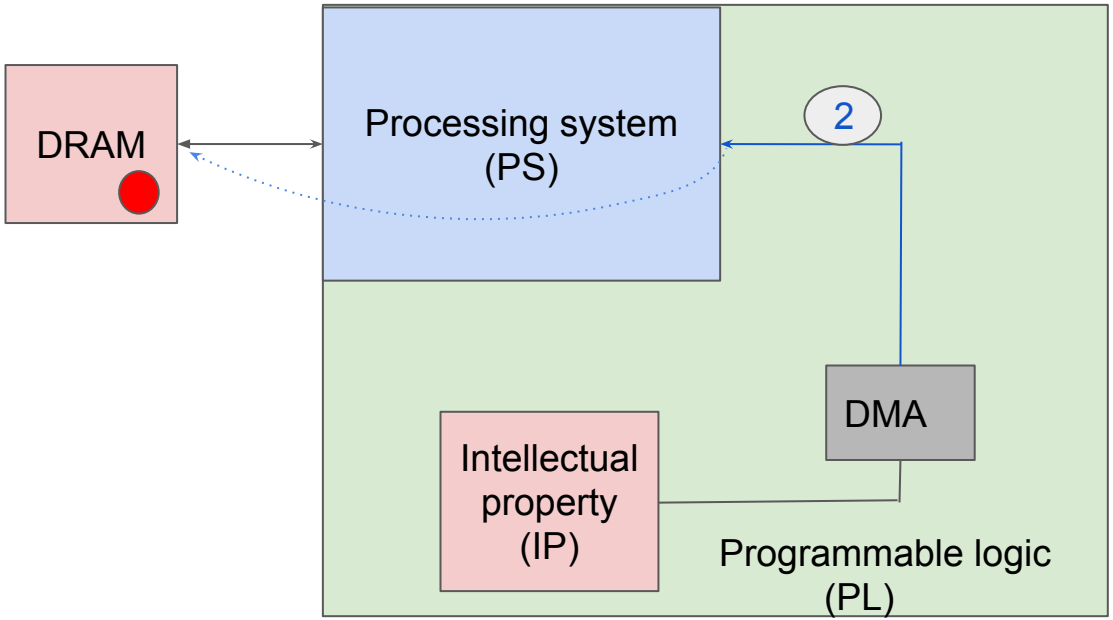
FPGA end-to-end flow of data



1) **PS -> Configure DMA AXI_LITE**

*DMA- Direct memory access

FPGA end-to-end flow of data

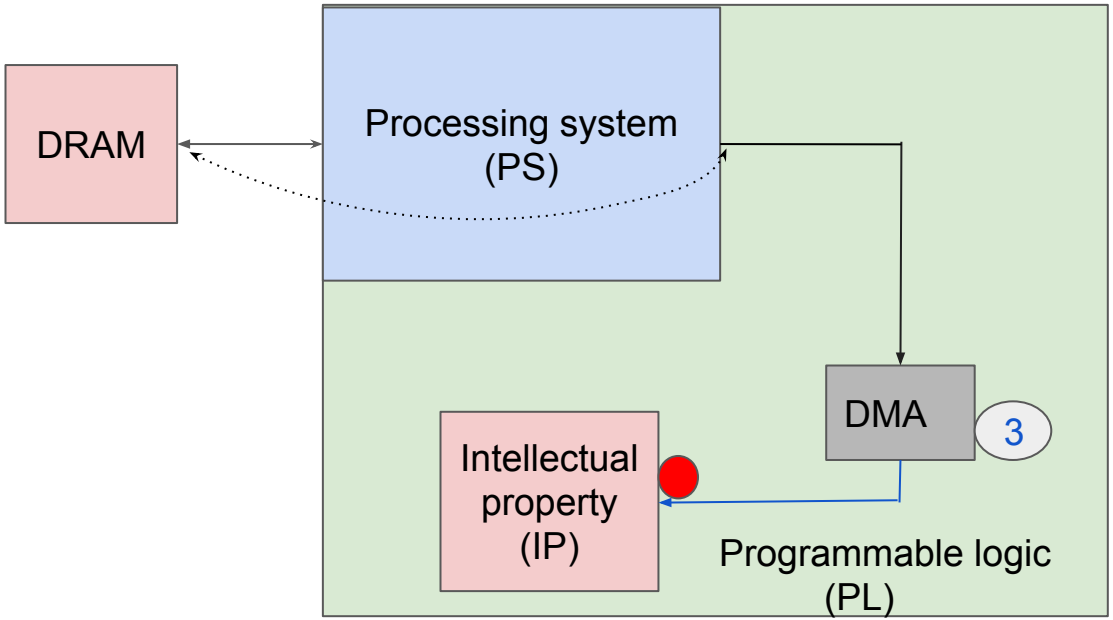


2) PS (&data) -> DMA

DMA will fetch the data from memory

*DMA- Direct memory access

FPGA end-to-end flow of data

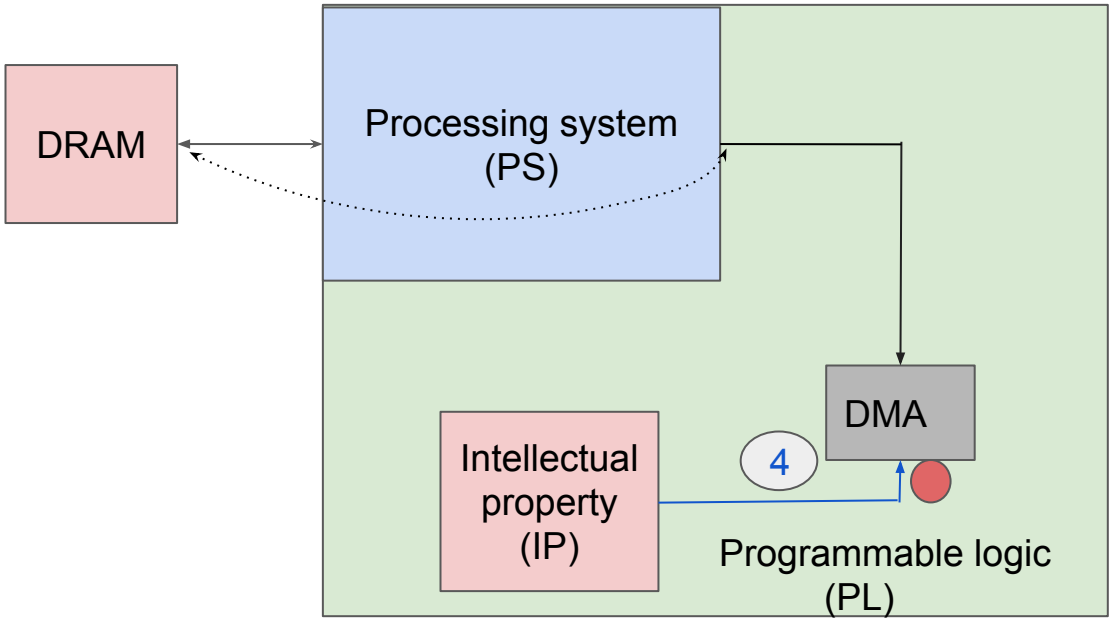


3) DMA(data) -> IP

DMA: Supply data to the IP

*DMA- Direct memory access

FPGA end-to-end flow of data

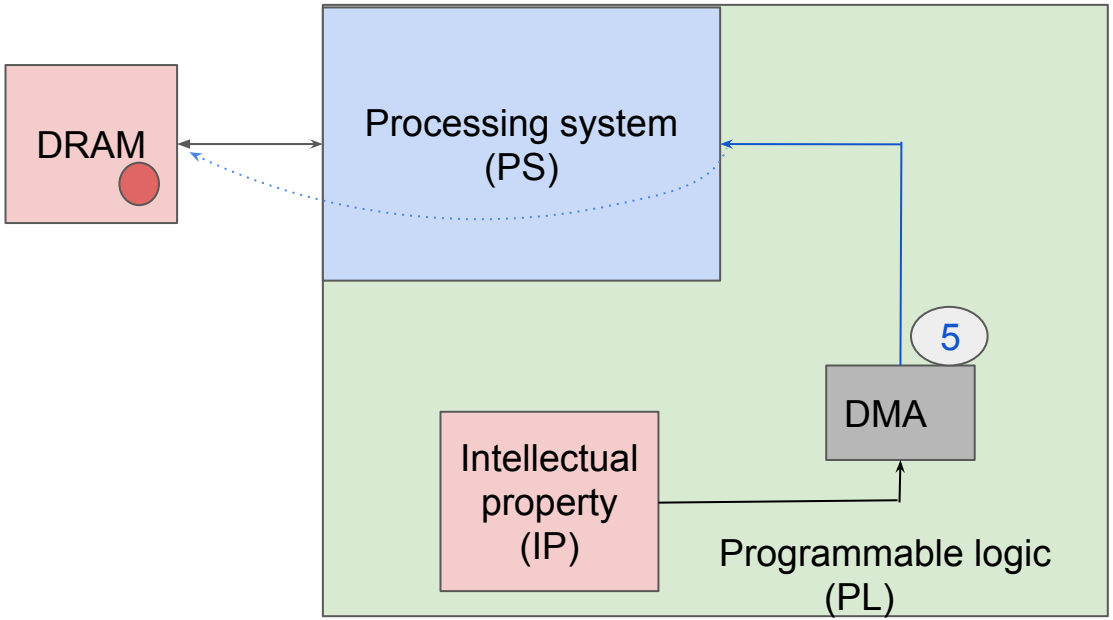


4) IP (data*) -> DMA

IP sent the data back to the DMA, after computation

*DMA- Direct memory access

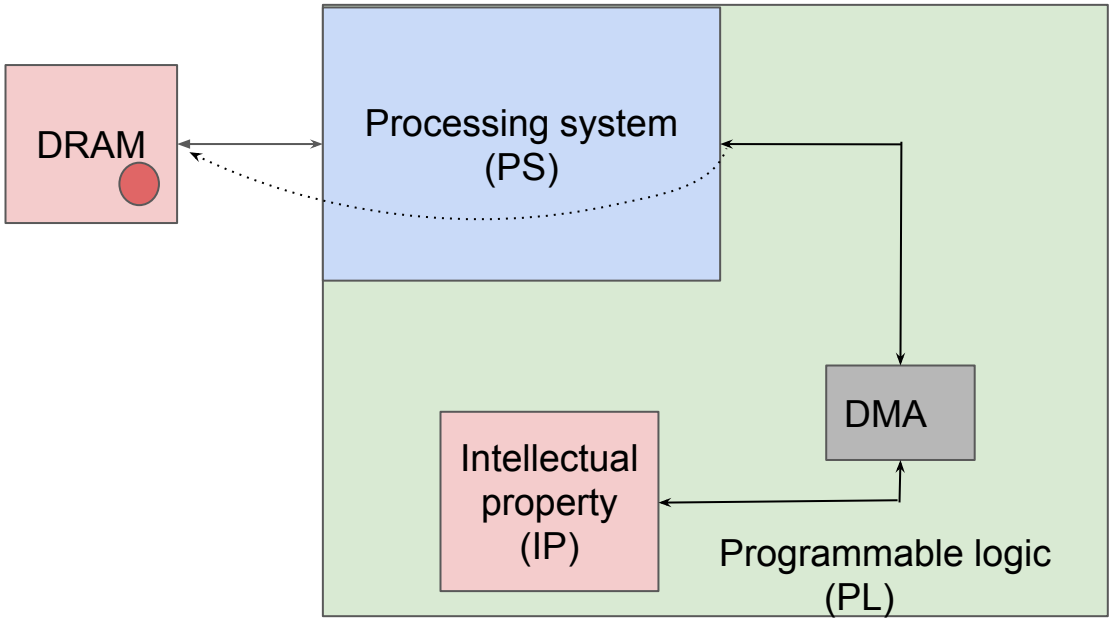
FPGA end-to-end flow of data



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

*DMA- Direct memory access

FPGA end-to-end flow of data

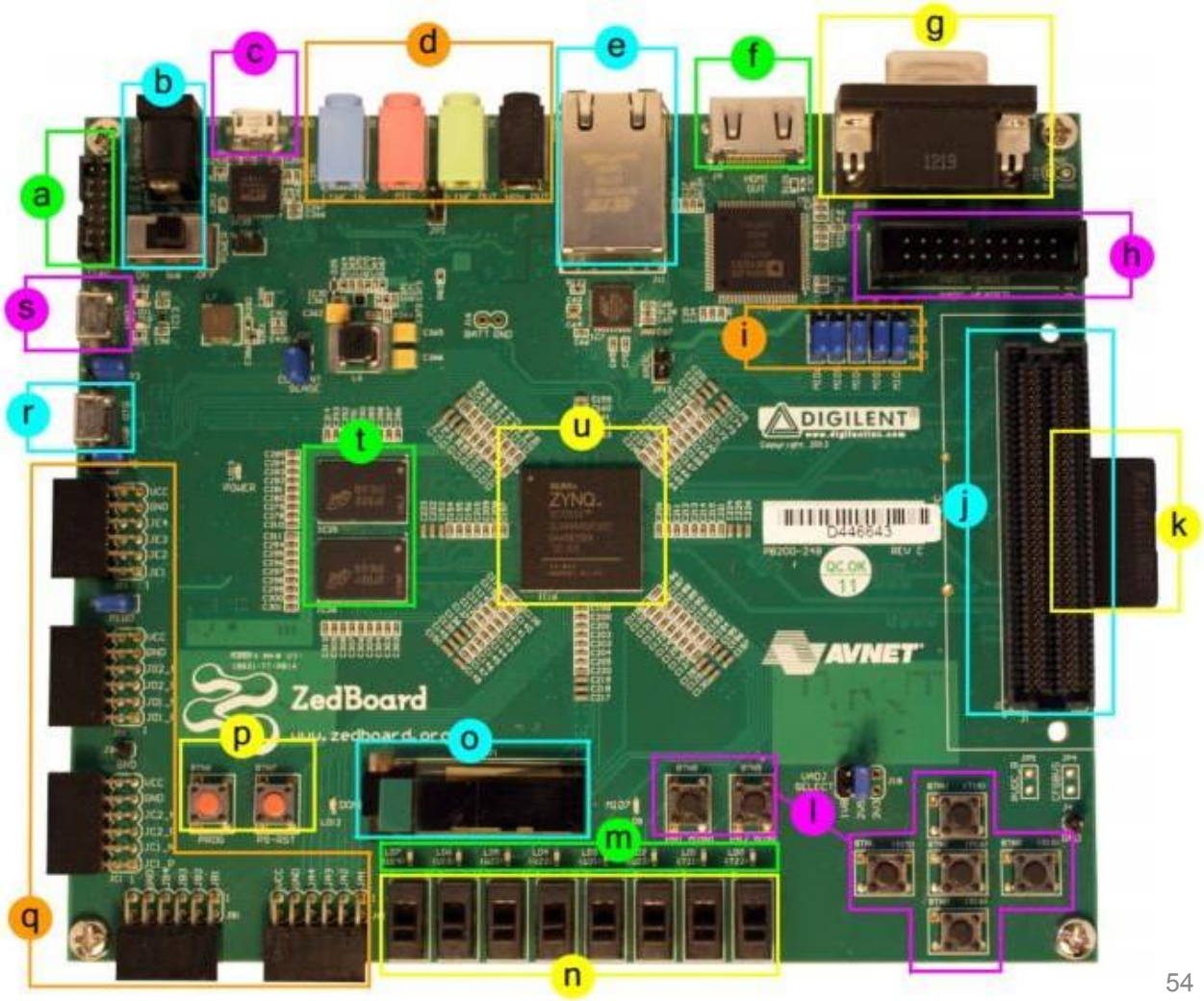


1) PS -> Configure DMA
2) PS (&data) -> DMA
3) DMA(data) -> IP
4) IP (data*) -> DMA
5) DMA (data)-> DRAM and inform the (&data) to PS.

*DMA- Direct memory access

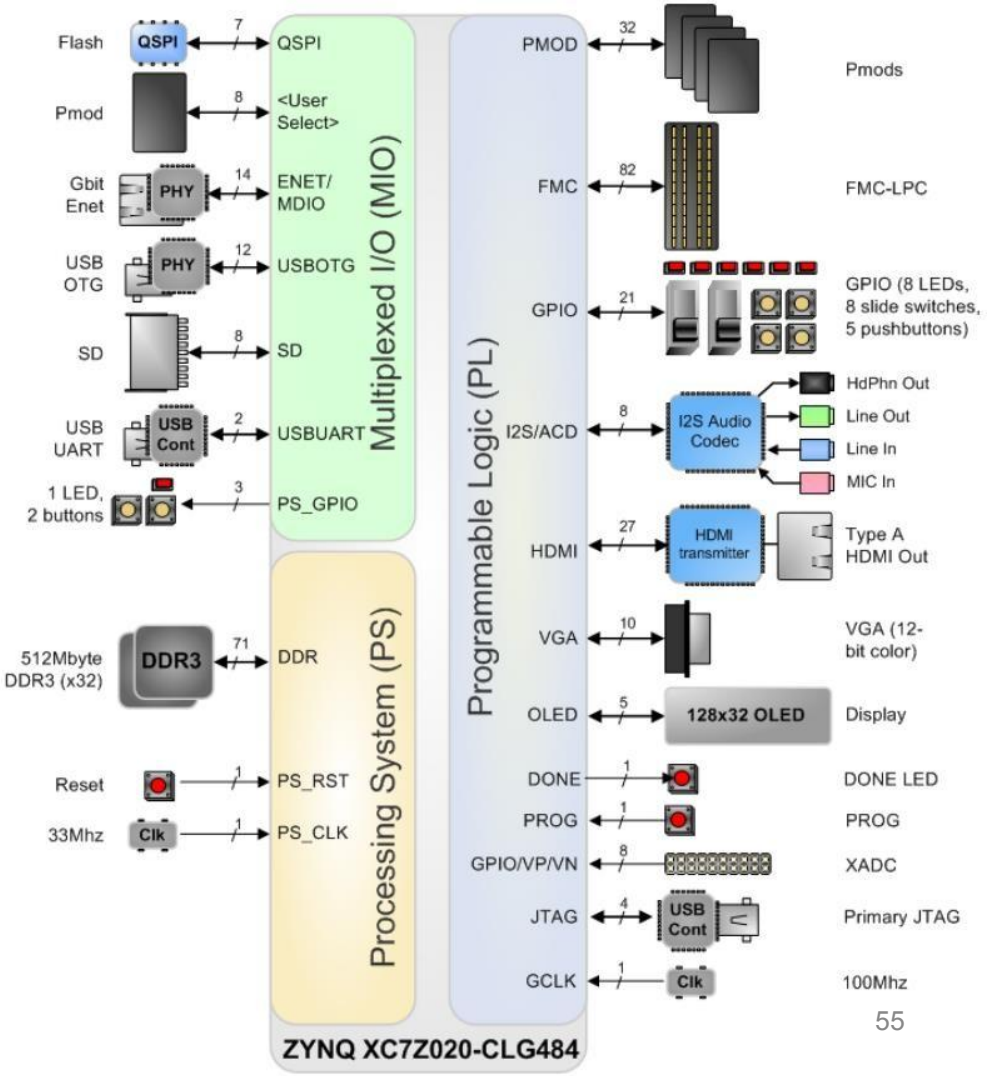
Zedboard

- 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)

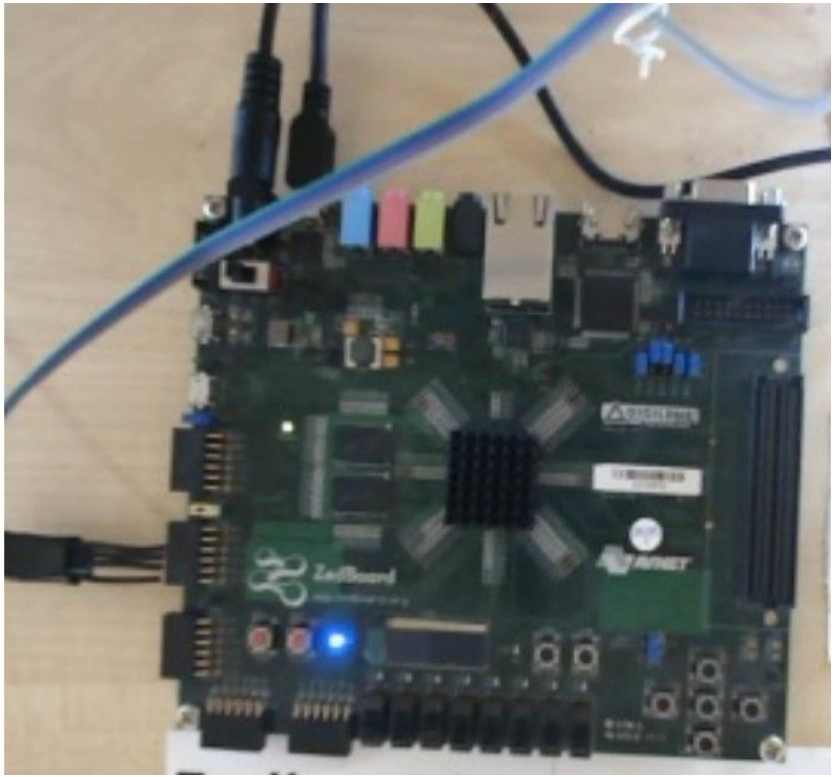


Zedboard

- 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
  - ○ 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:

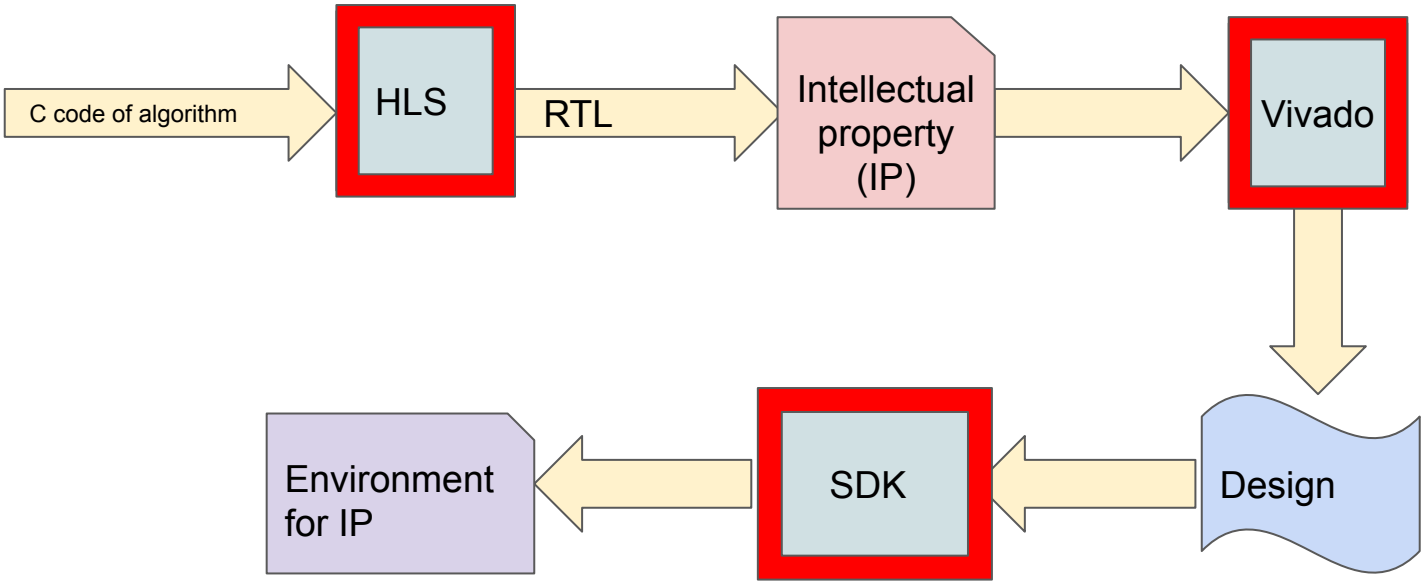
  $ git clone https://github.com/pnl-iiitd/2023-P4-for-all-tutorial.git
- $ source /tools/Xilinx/Vivado/2019.1/settings64.sh
- $ vivado &
- $ vivado_hls

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

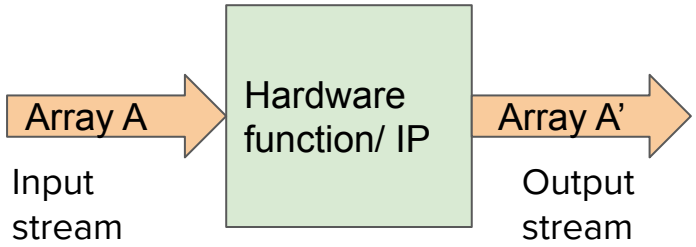
Restriction in HLS

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>

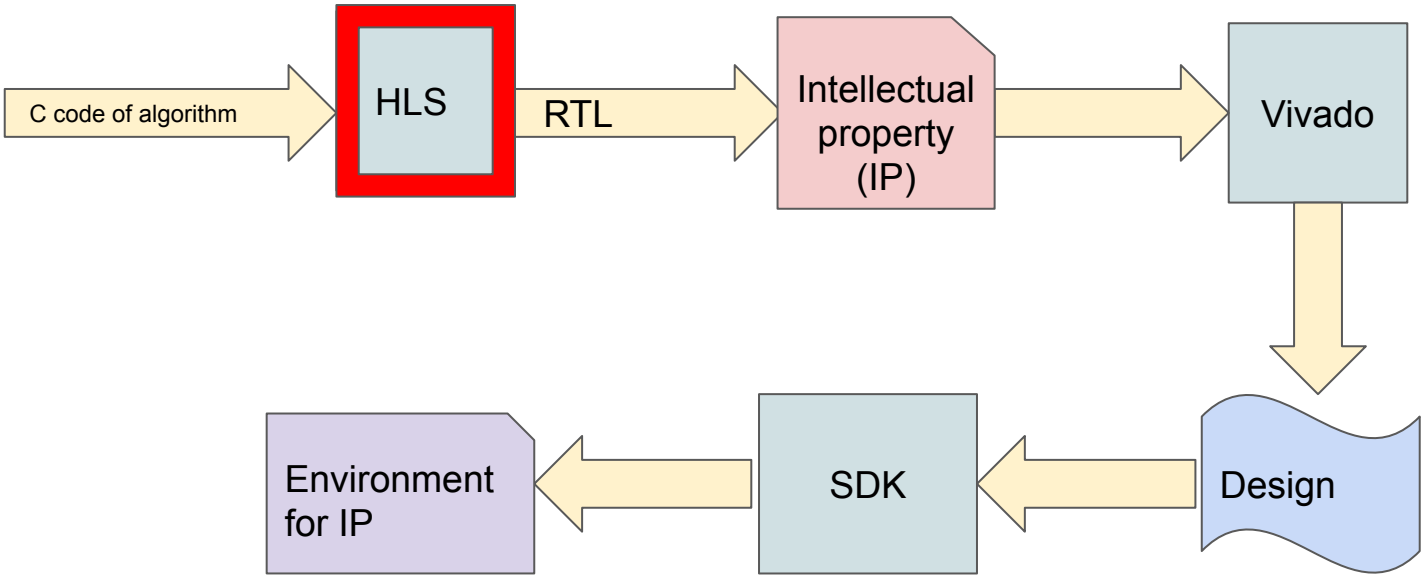
FPGA hands-on

Square of number

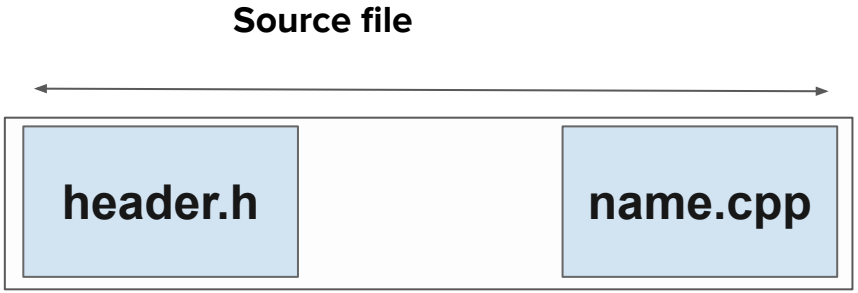
- Input: An array of numbers say 1 to 10 i.e 1, 2, 3, 4, 5, 6, 7, 8, 9.
- Output: An array of square of given numbers i.e 1, 4,9, 16, 25, 36, 49, 64, 81.

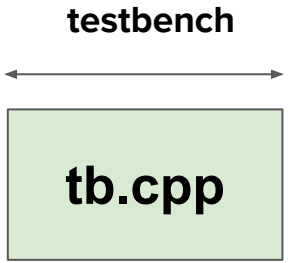


FPGA: Workflow



HLS: Higher level synthesis

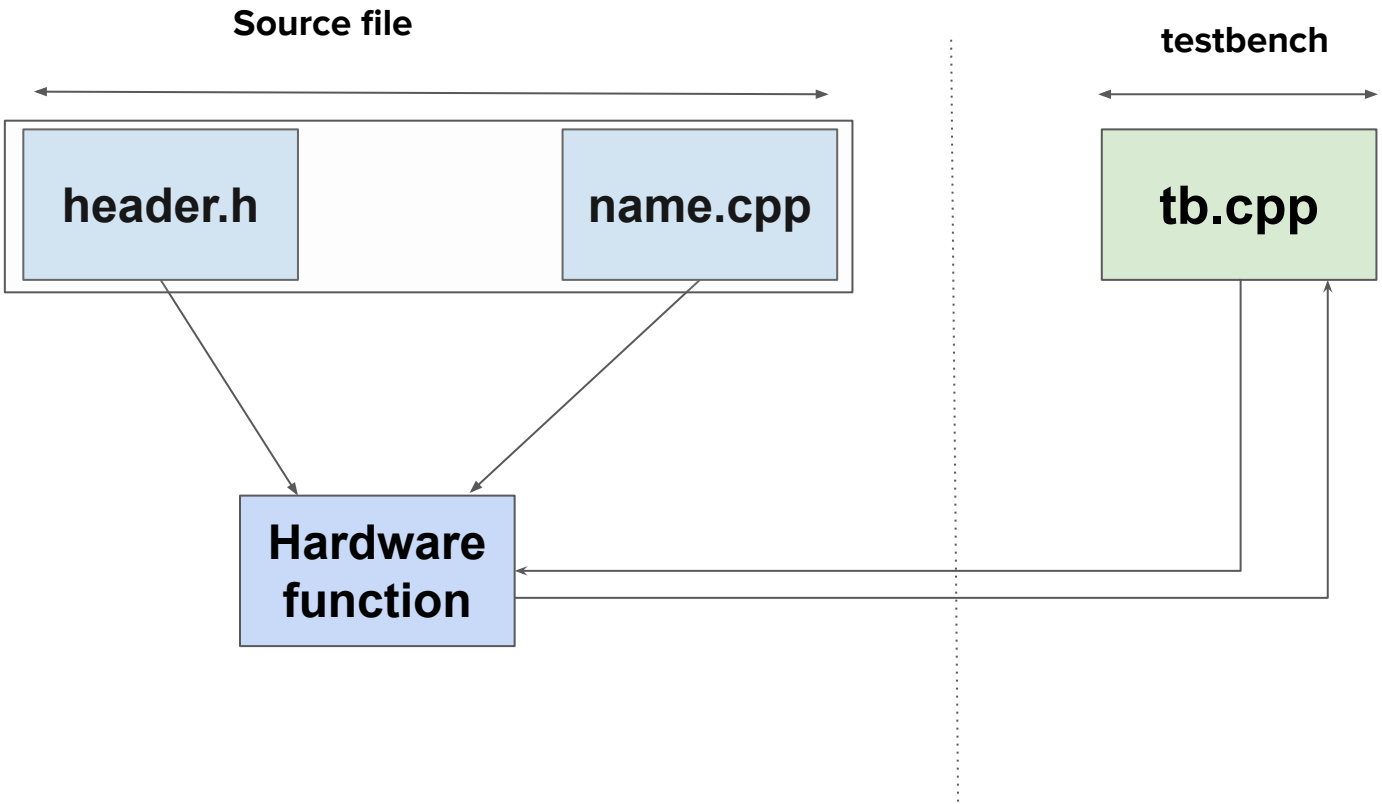




1) Main function
2) Call the hardware function

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

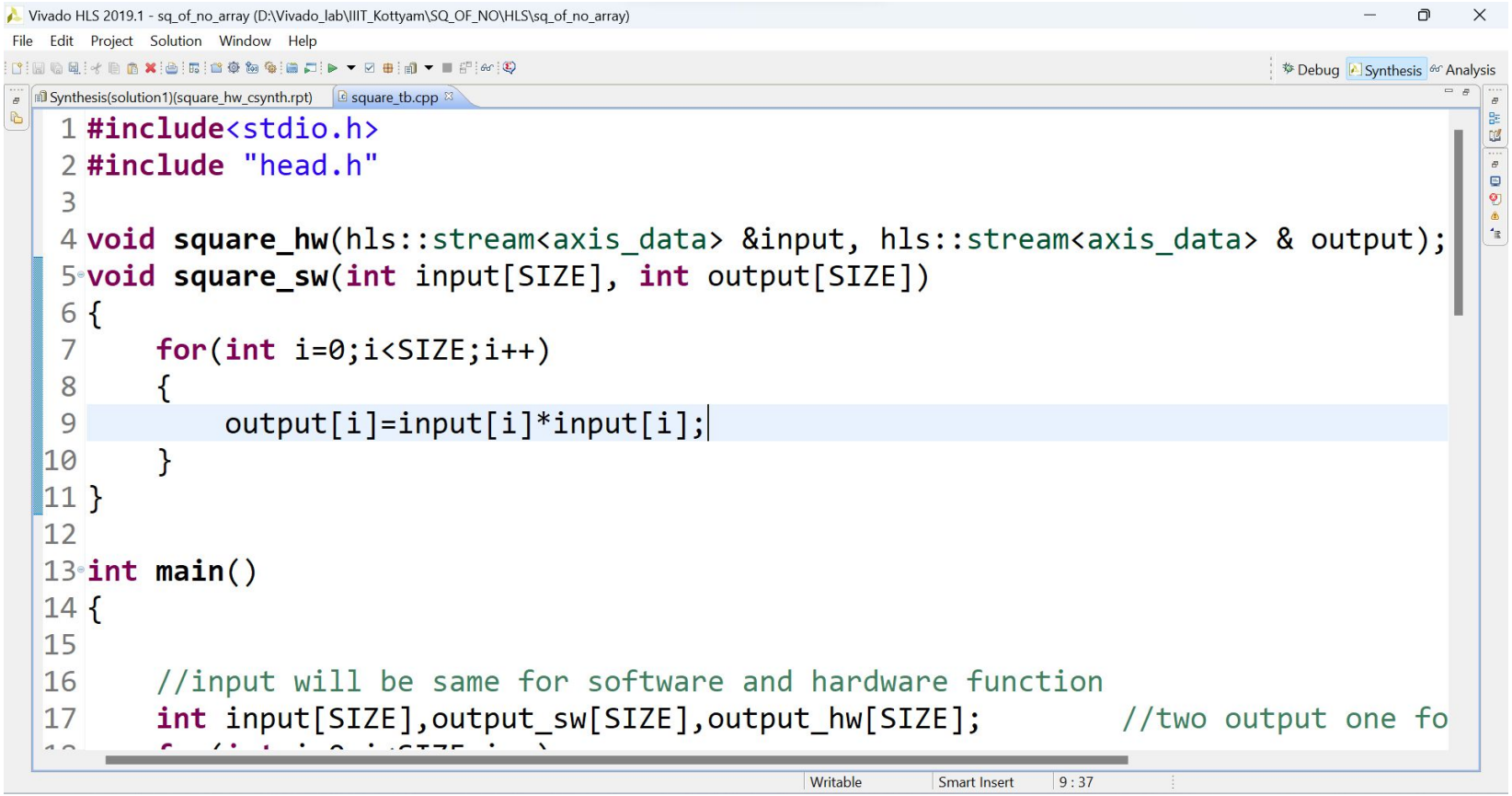
HLS: Square of number



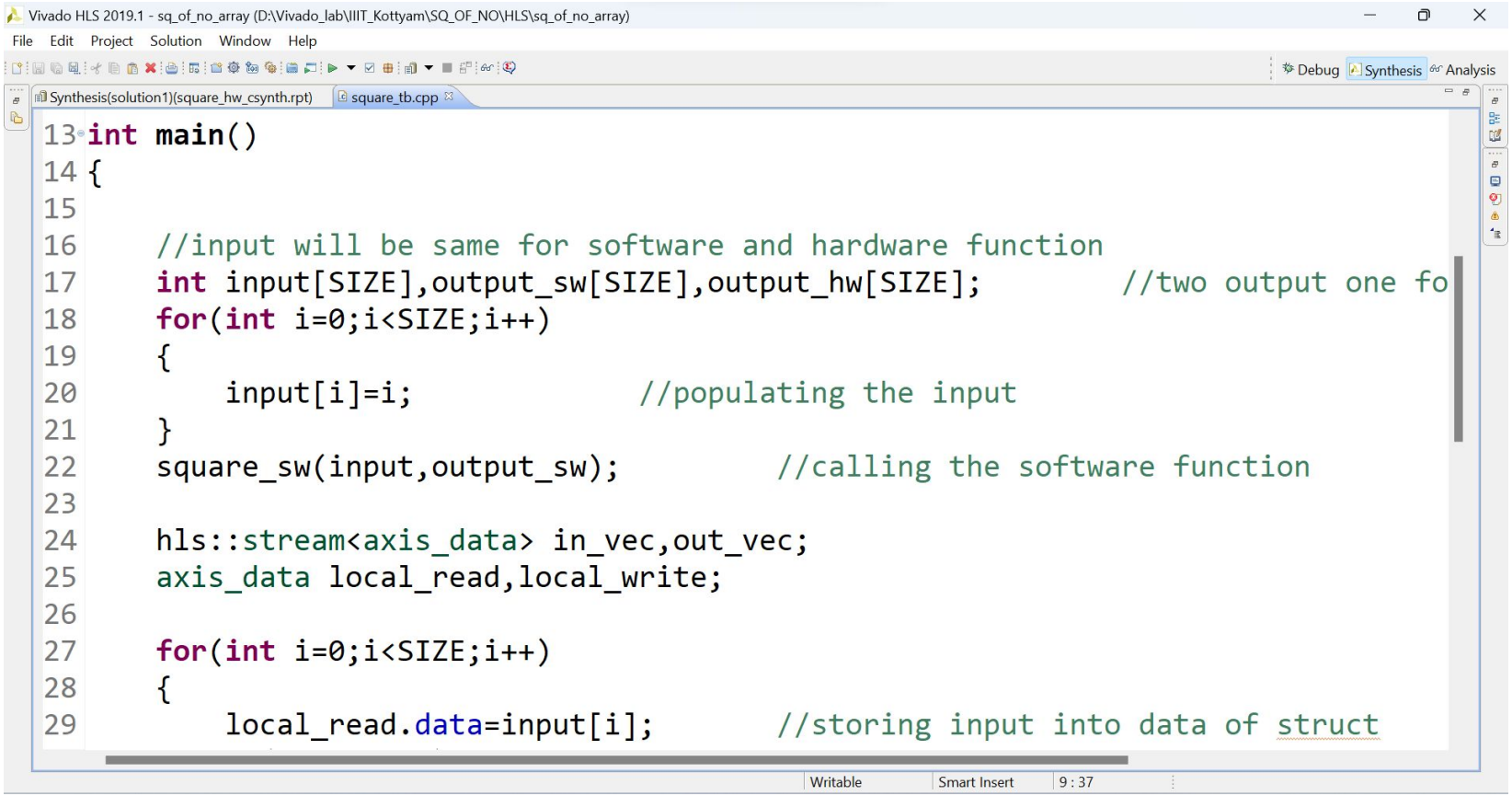
HLS: Square of number

```
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
```

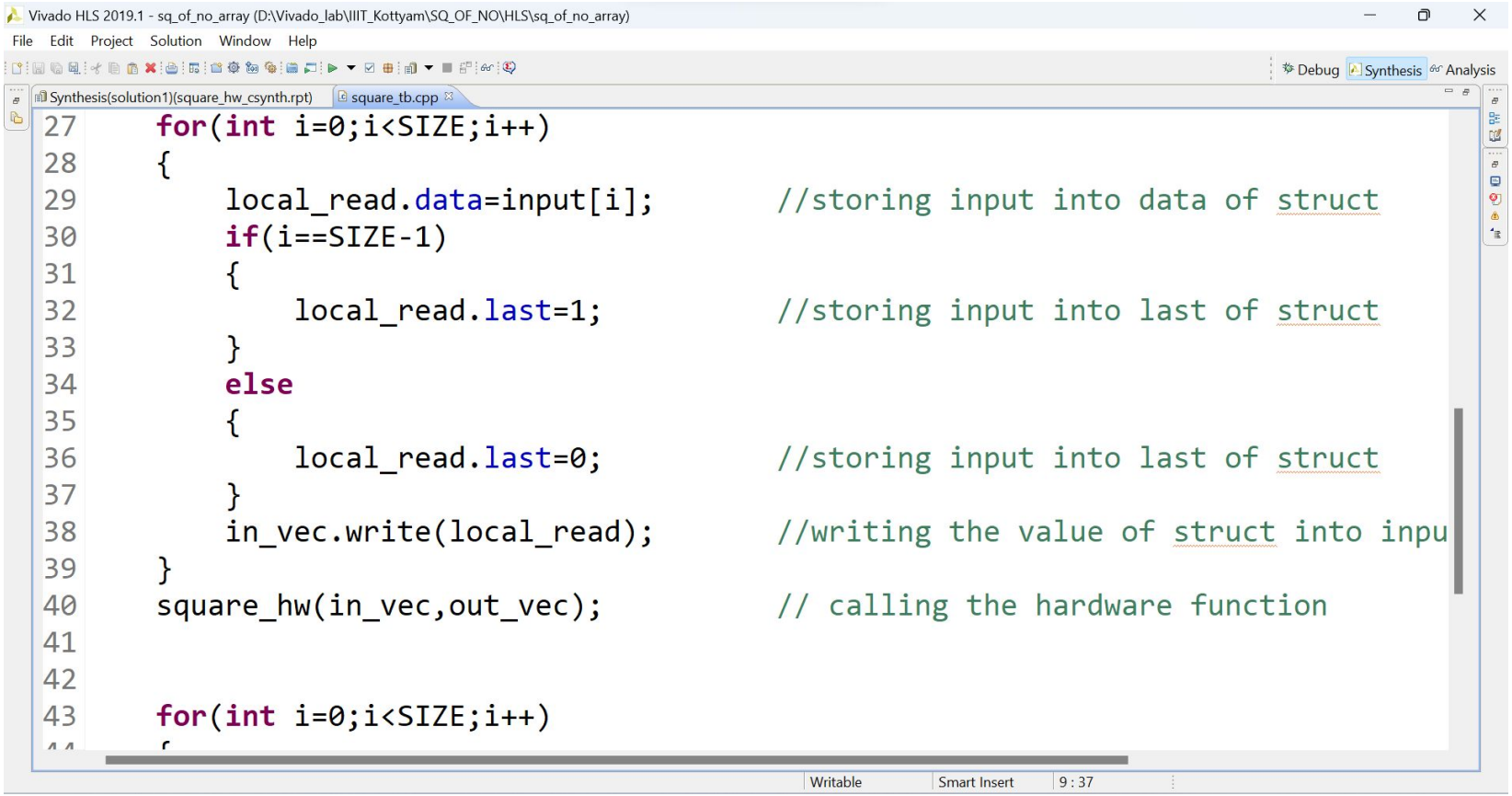
HLS: Square of number



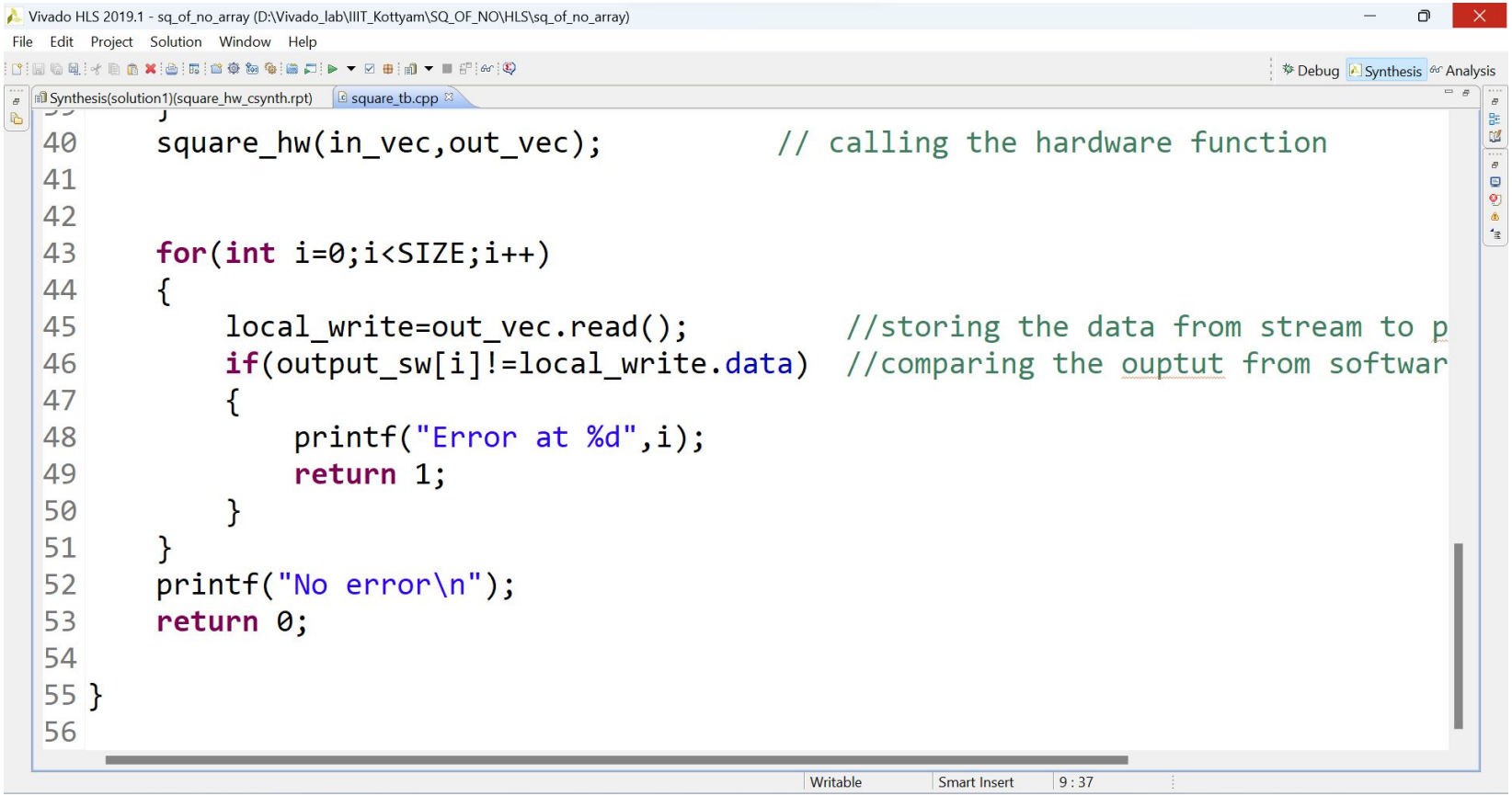
HLS: Square of number



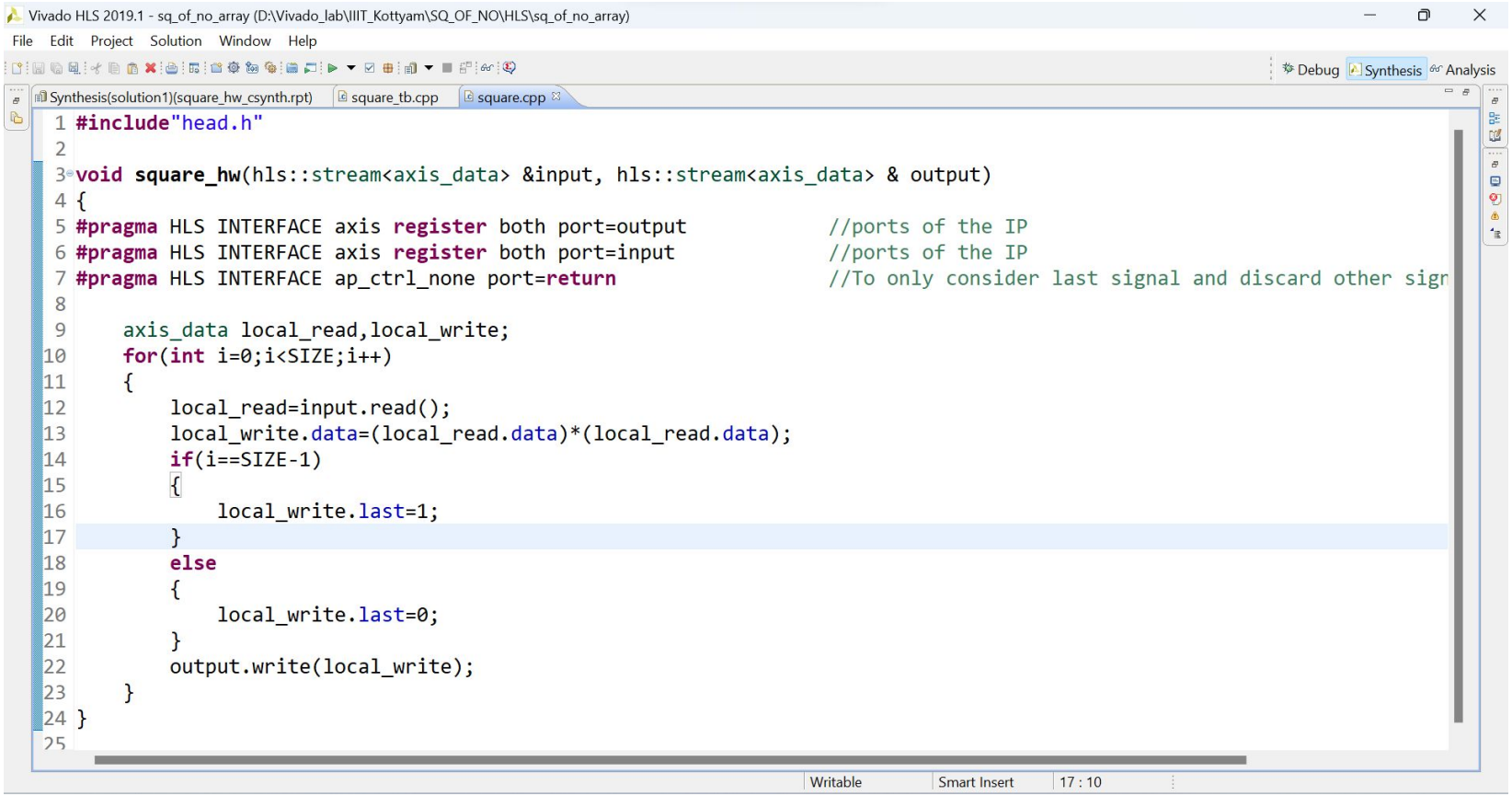
HLS: Square of number



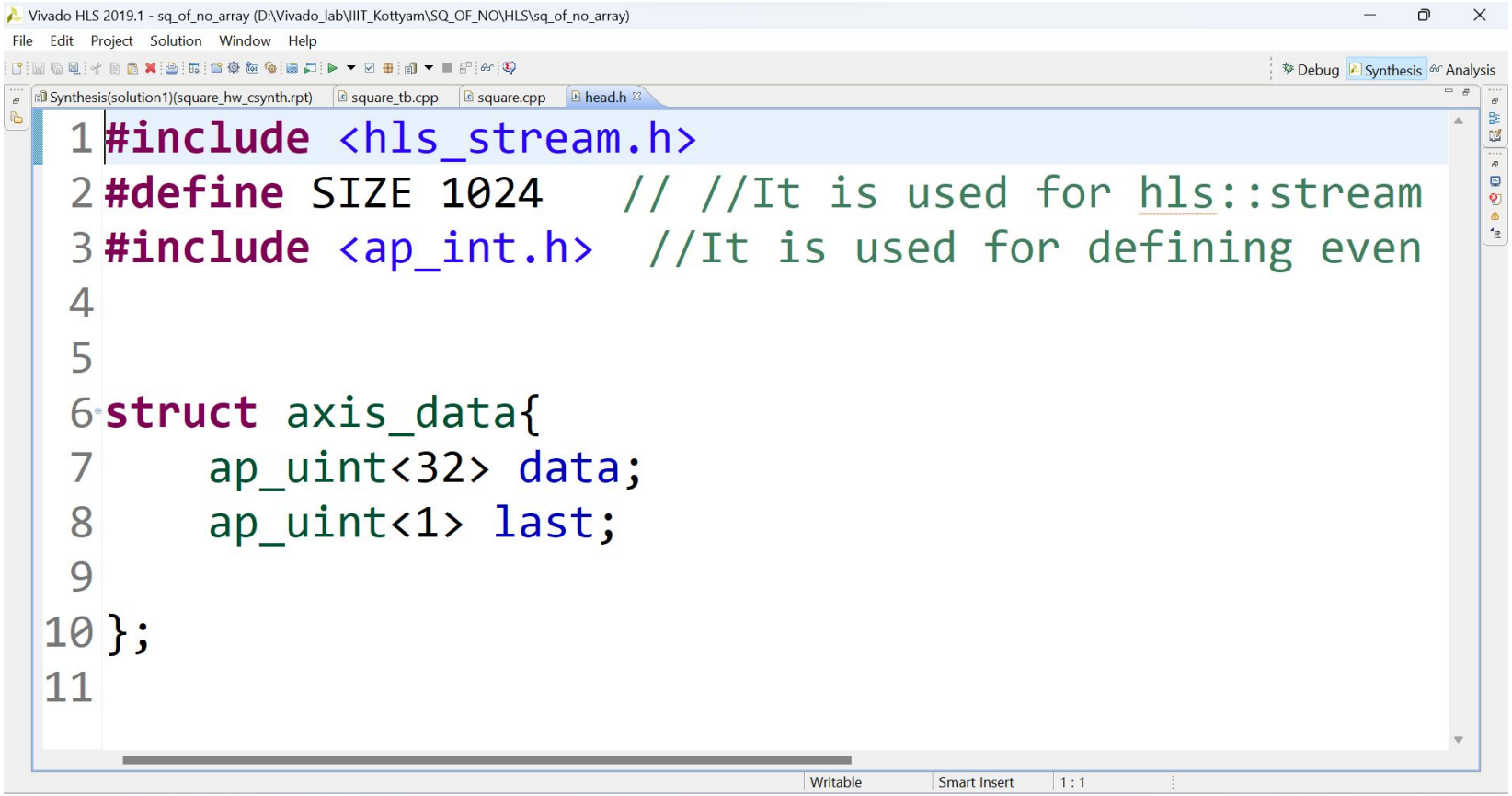
HLS: Square of number



HLS: Square of number



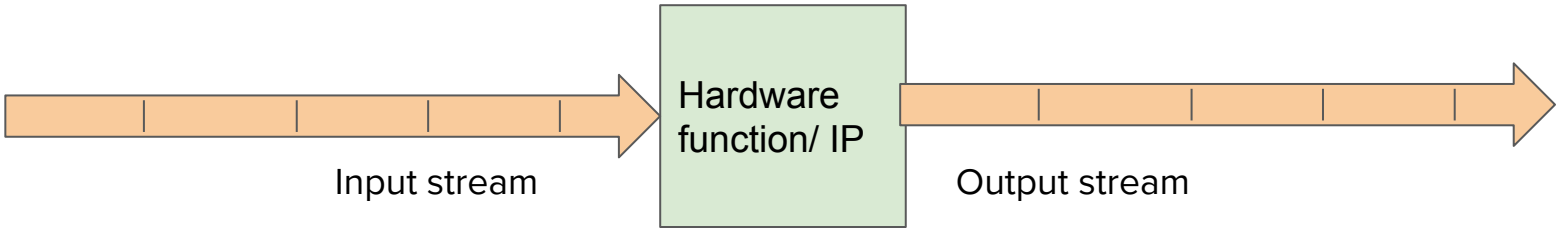
HLS: Square of number



HLS : Stream size

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

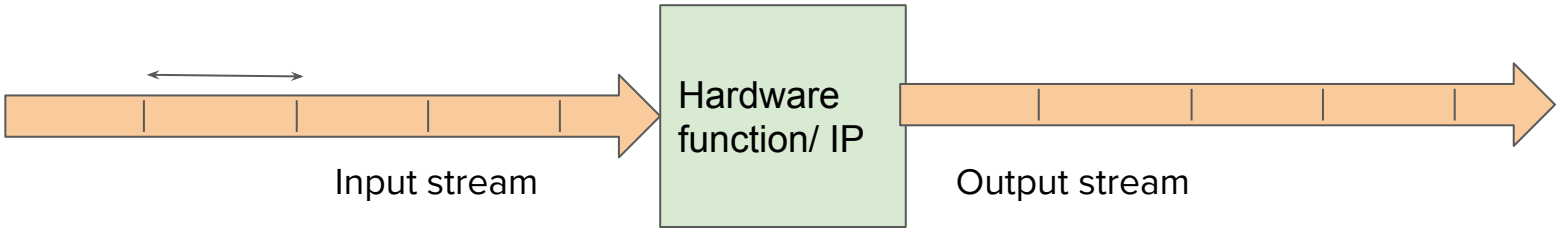
- The maximum amount of data can be transfer in 1 iteration.
- Stream works in FIFO manner



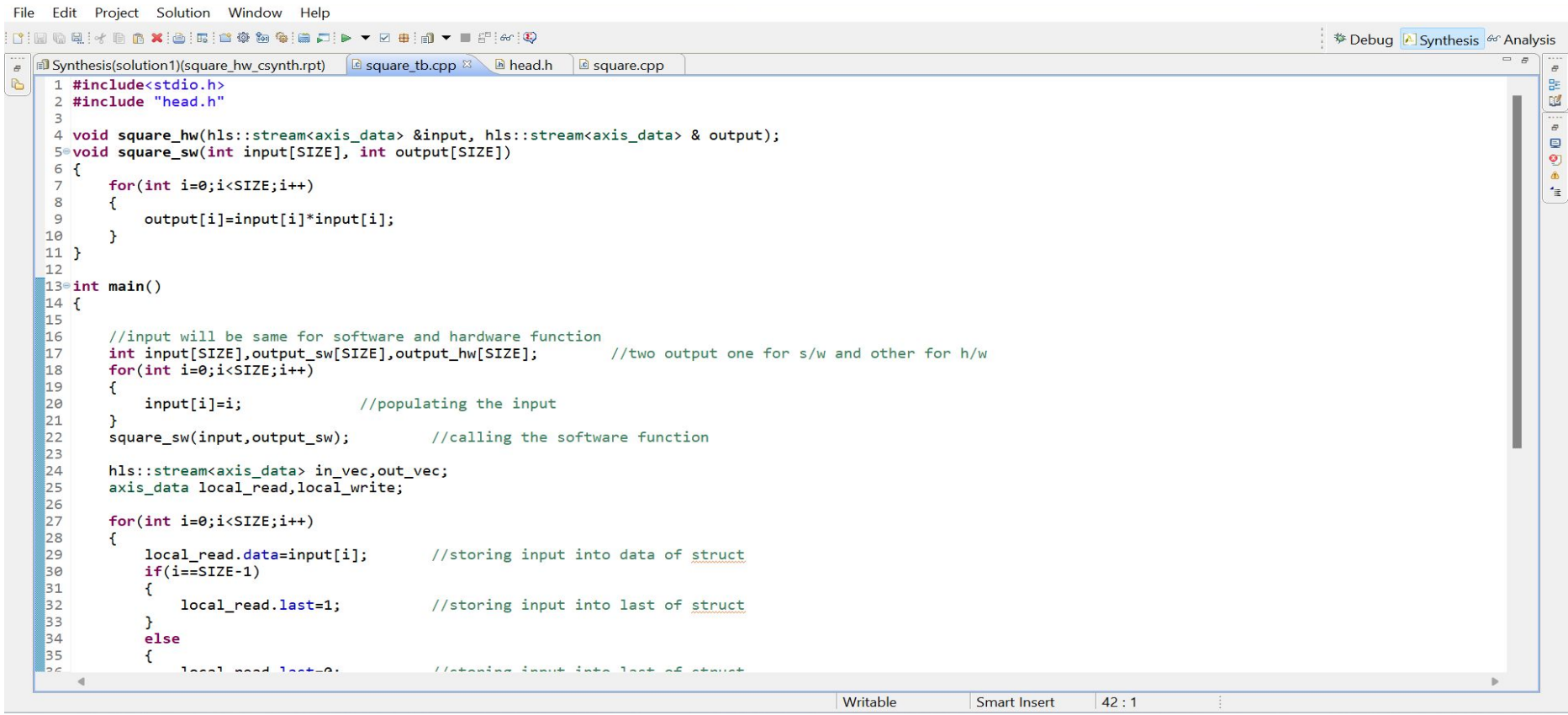
HLS : Stream size

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

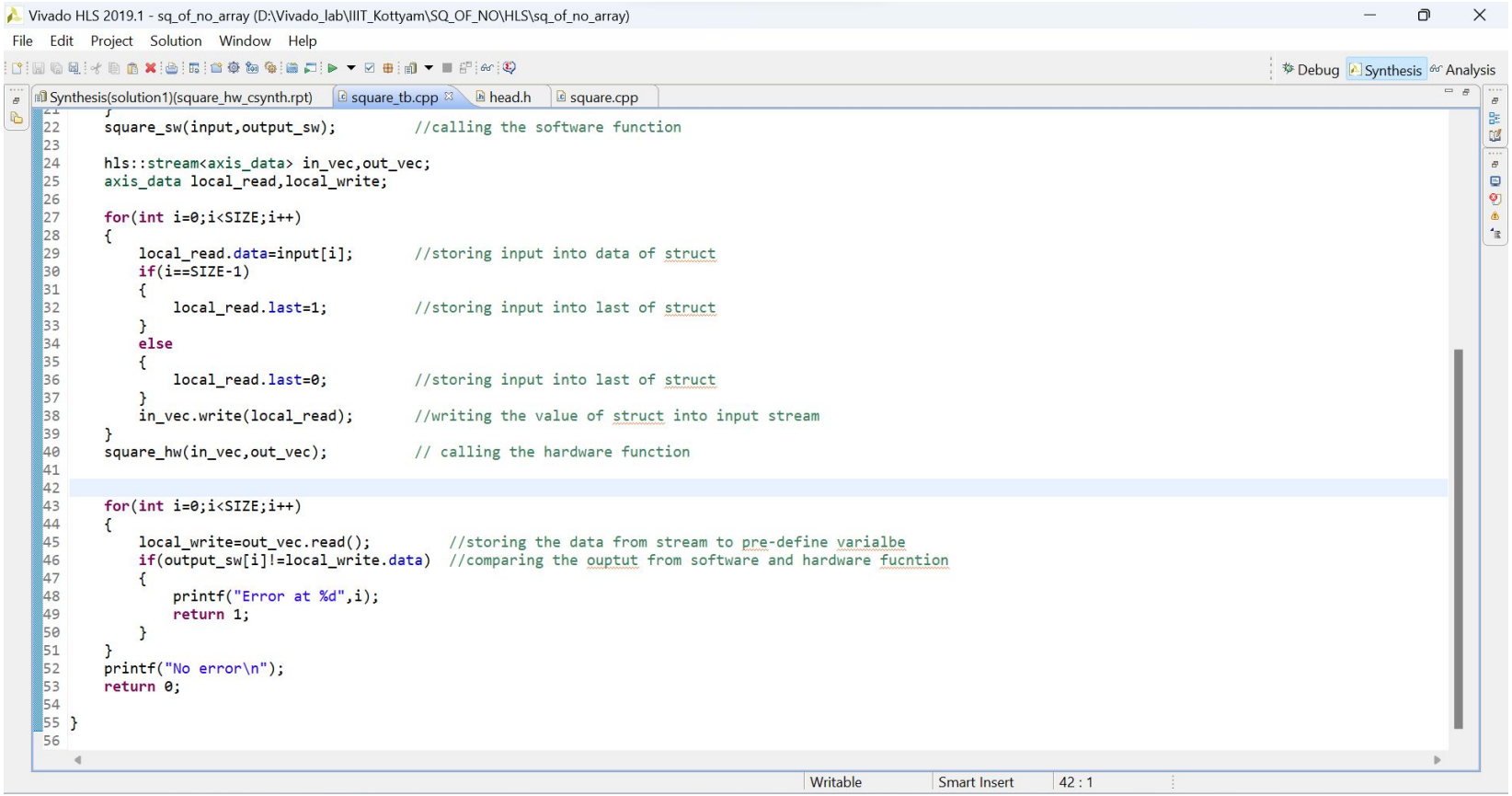
- The maximum amount of data can be transfer in 1 iteration.
- Stream works in FIFO manner



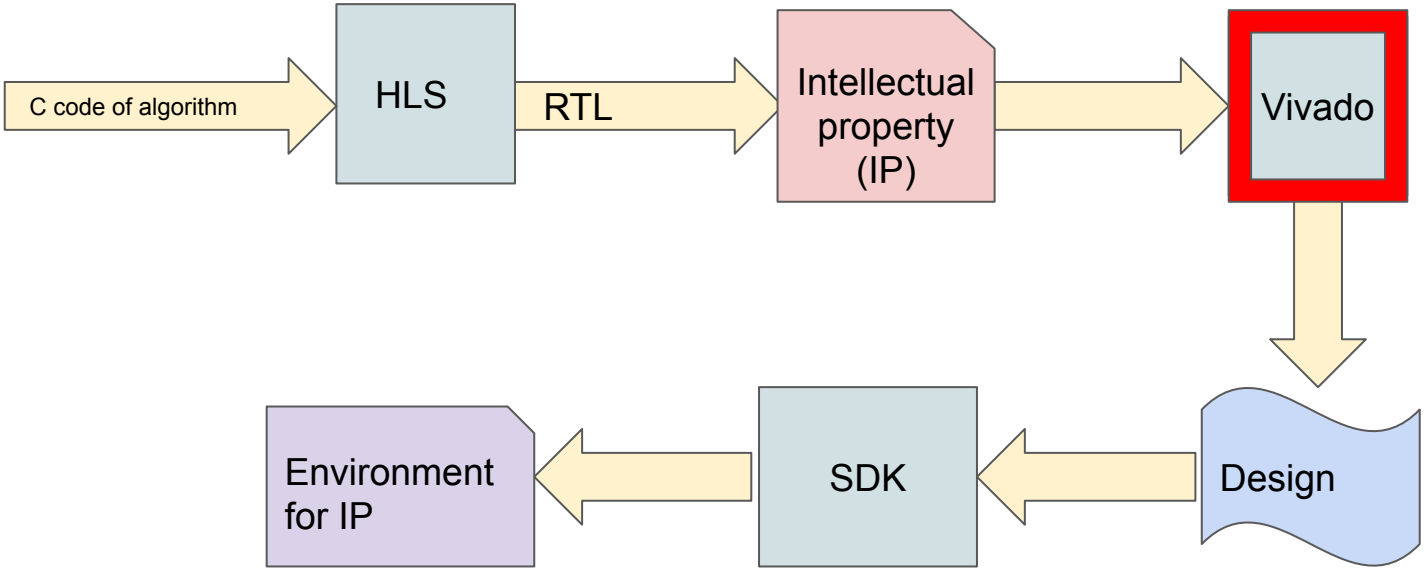
HLS: Square of number



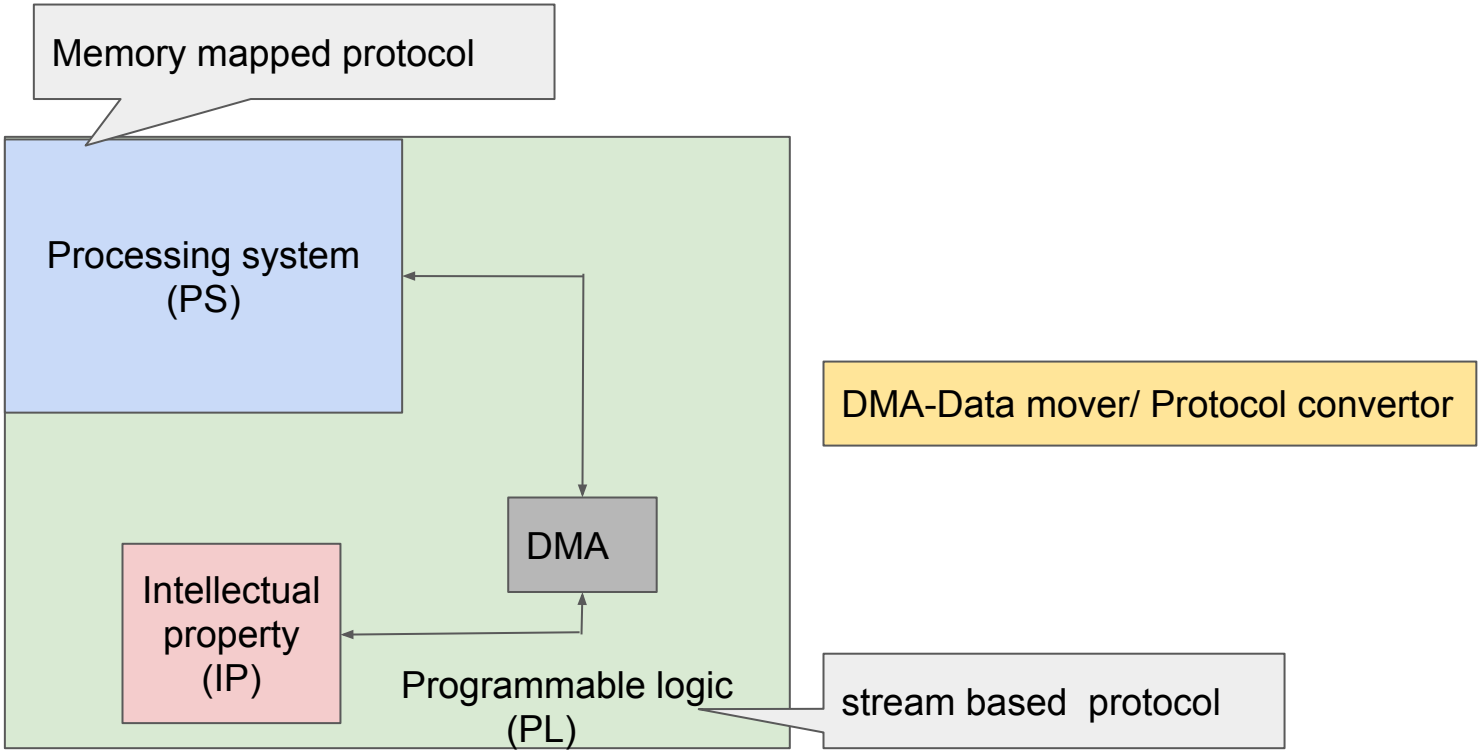
HLS: Square of number



FPGA: Workflow

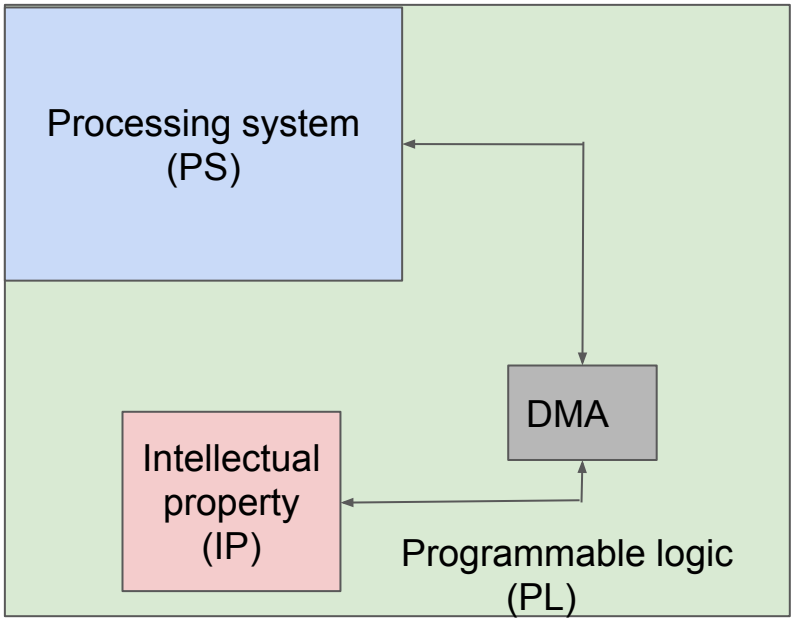


FPGA end-to-end working



*DMA- Direct memory access

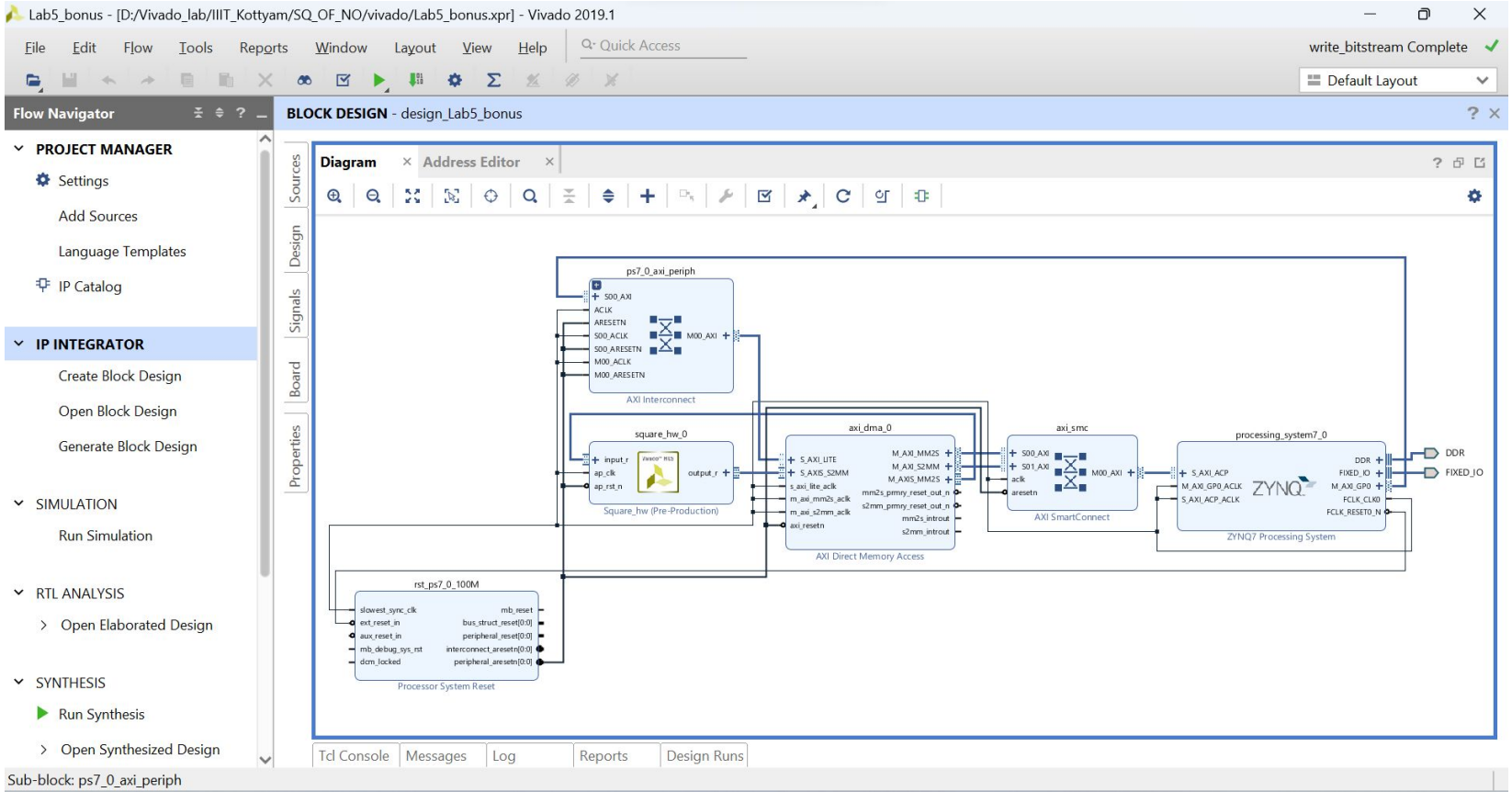
FPGA end-to-end working



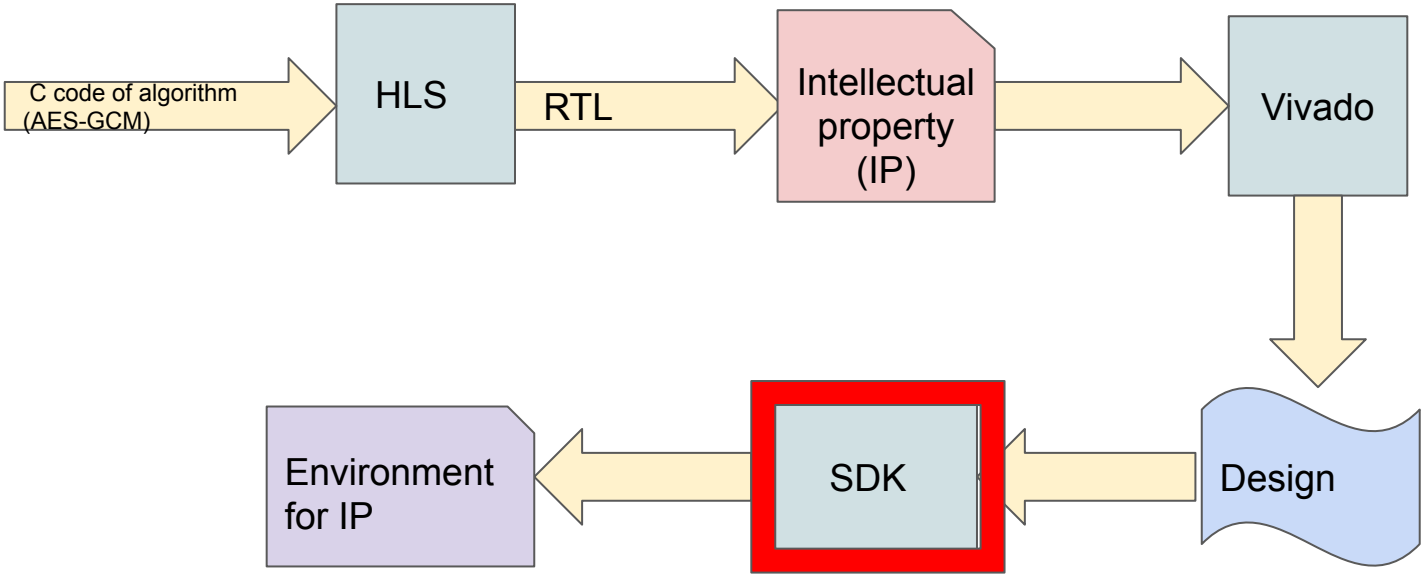
1) Master and slave
2) Master is always initiator of the transaction.
3) Transaction can be read/write

*DMA- Direct memory access

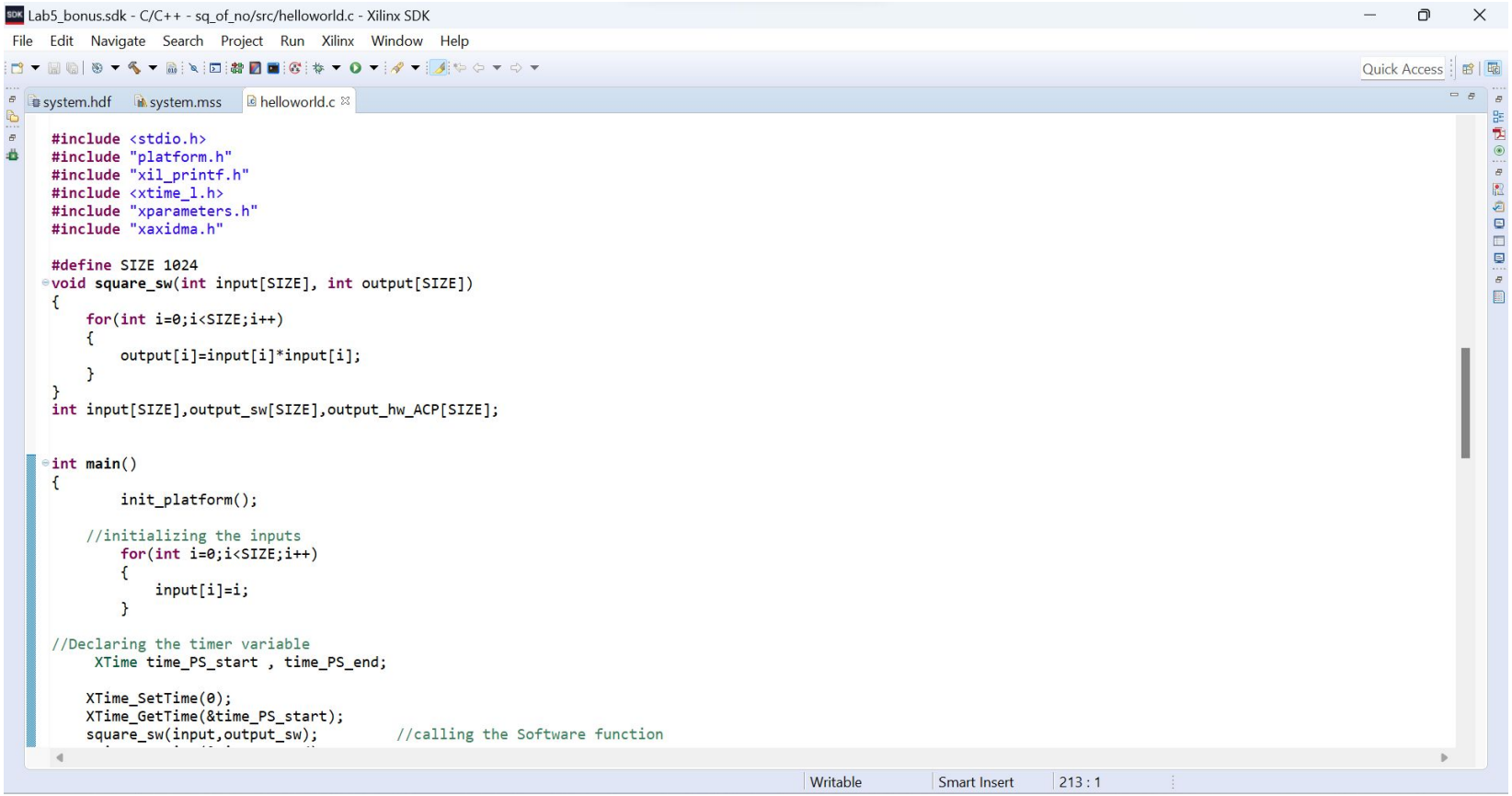
Vivado : Creation of block diagram



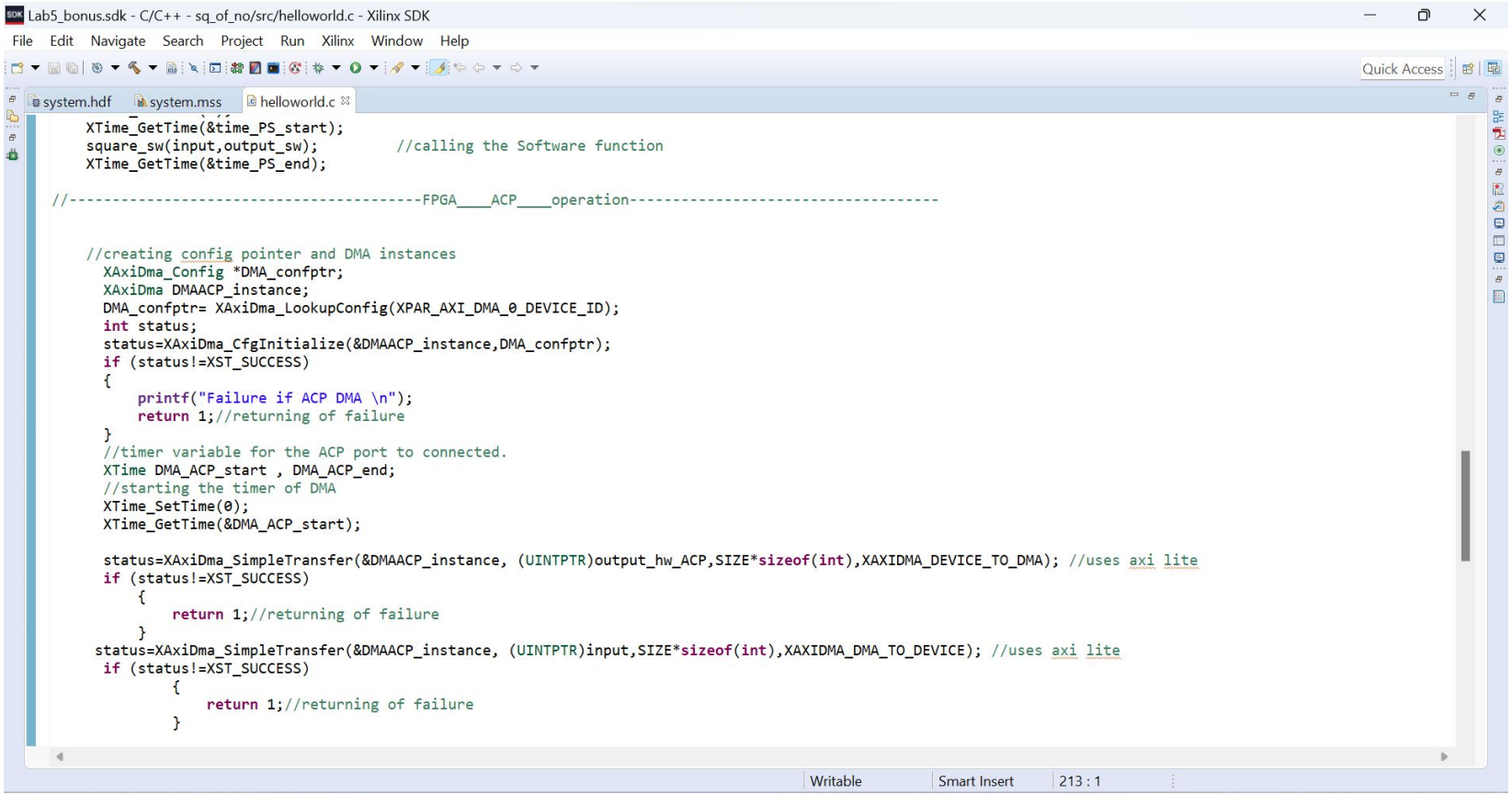
FPGA: Workflow



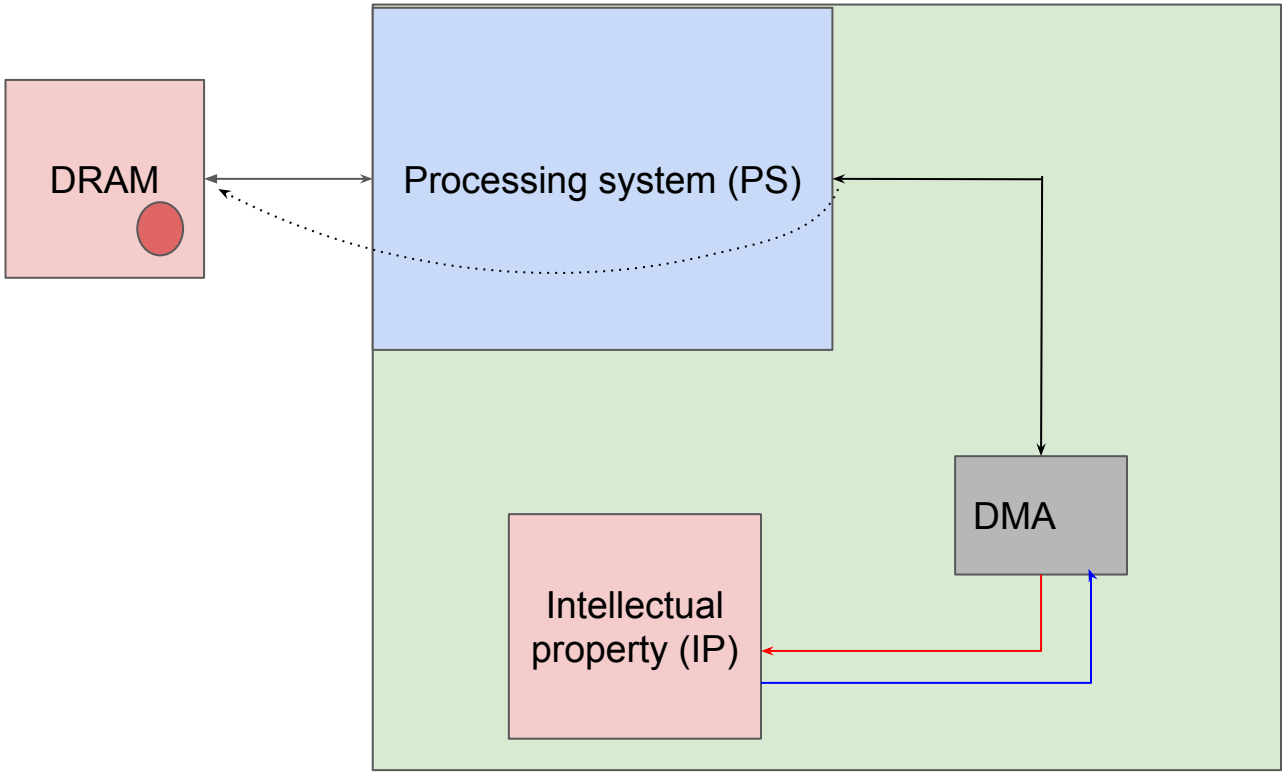
SDK


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_l.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


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
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
}
Writable
Smart Insert
213 : 1

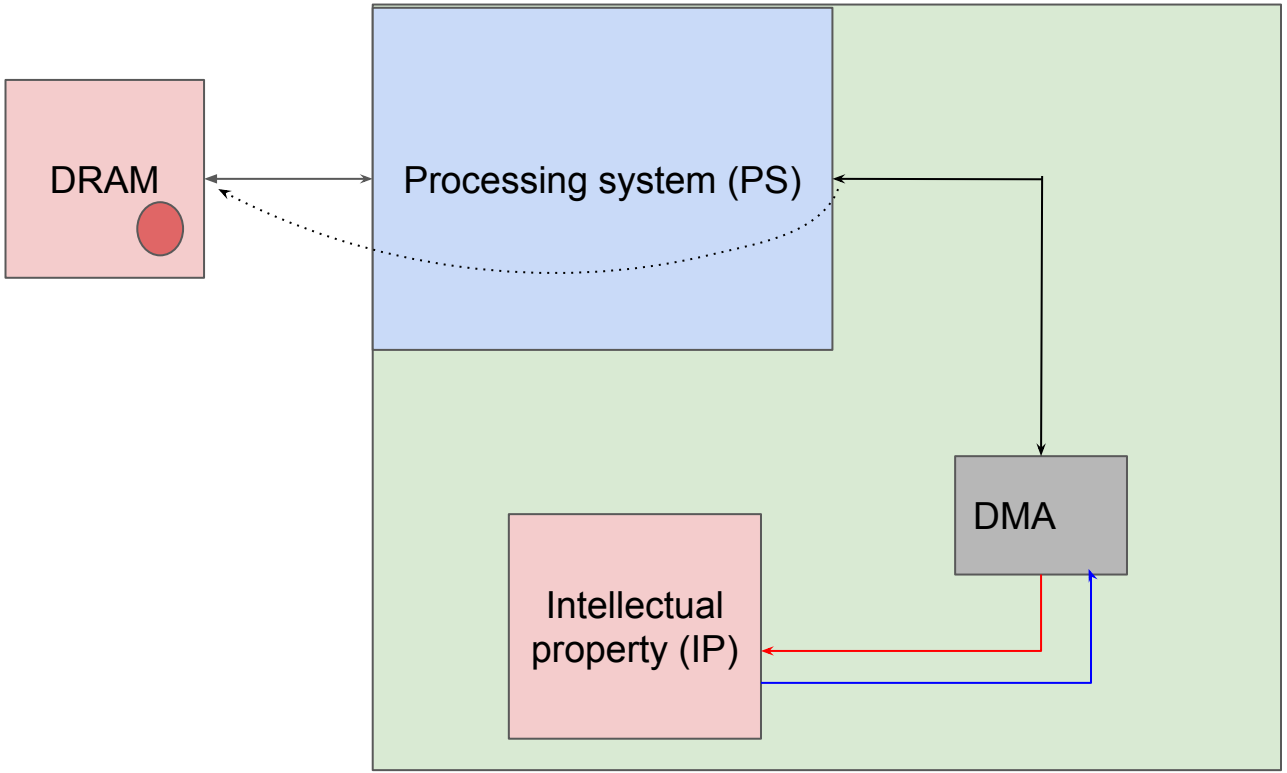
Link from DMA to IP and IP to DMA



Which link to setup first?

*DMA- Direct memory access

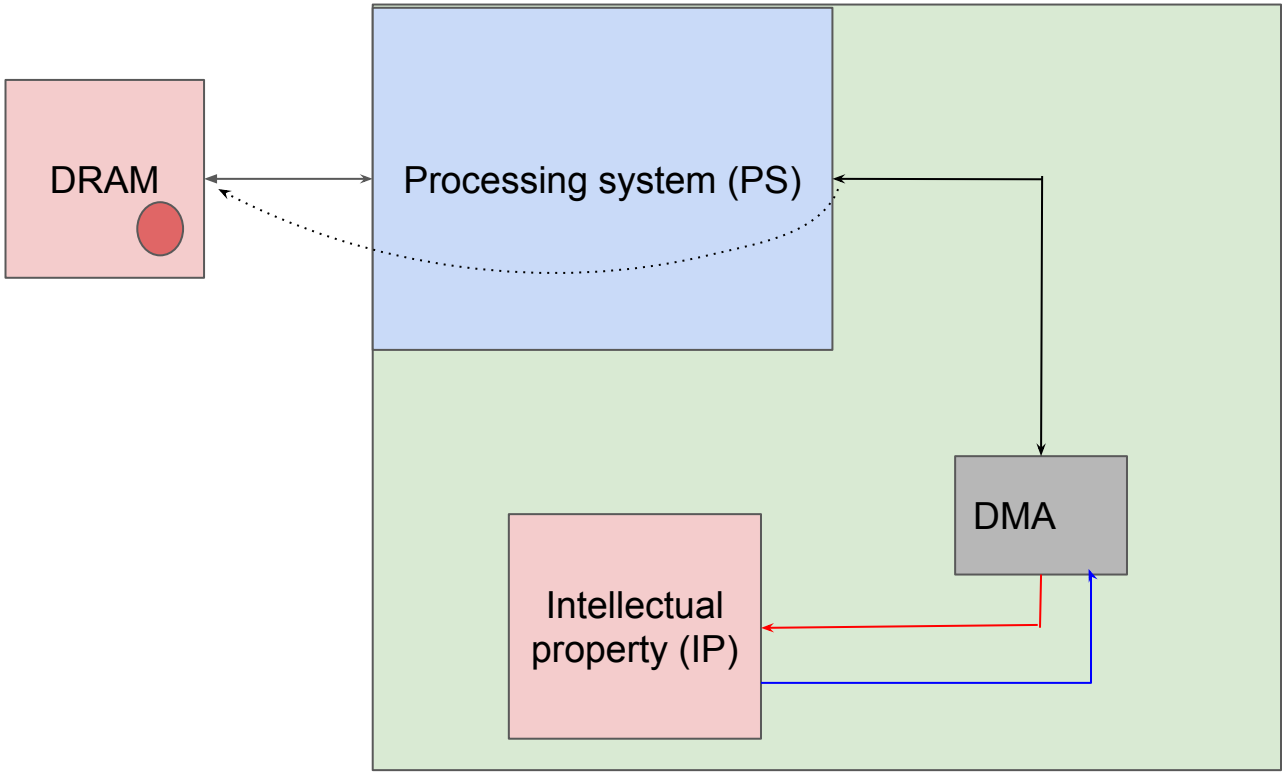
Link from DMA to IP and IP to DMA



IP to DMA

*DMA- Direct memory access

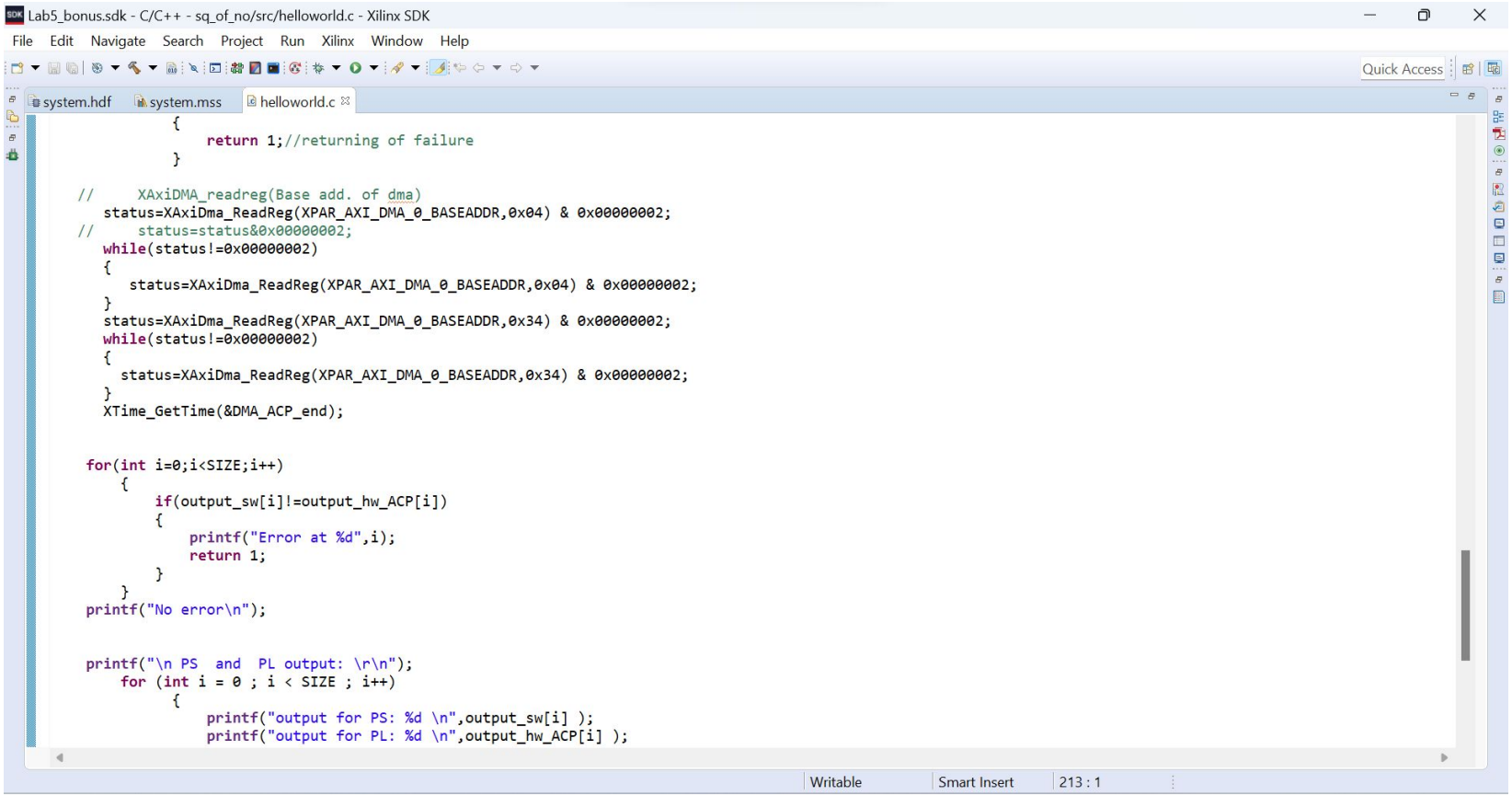
Back pressure



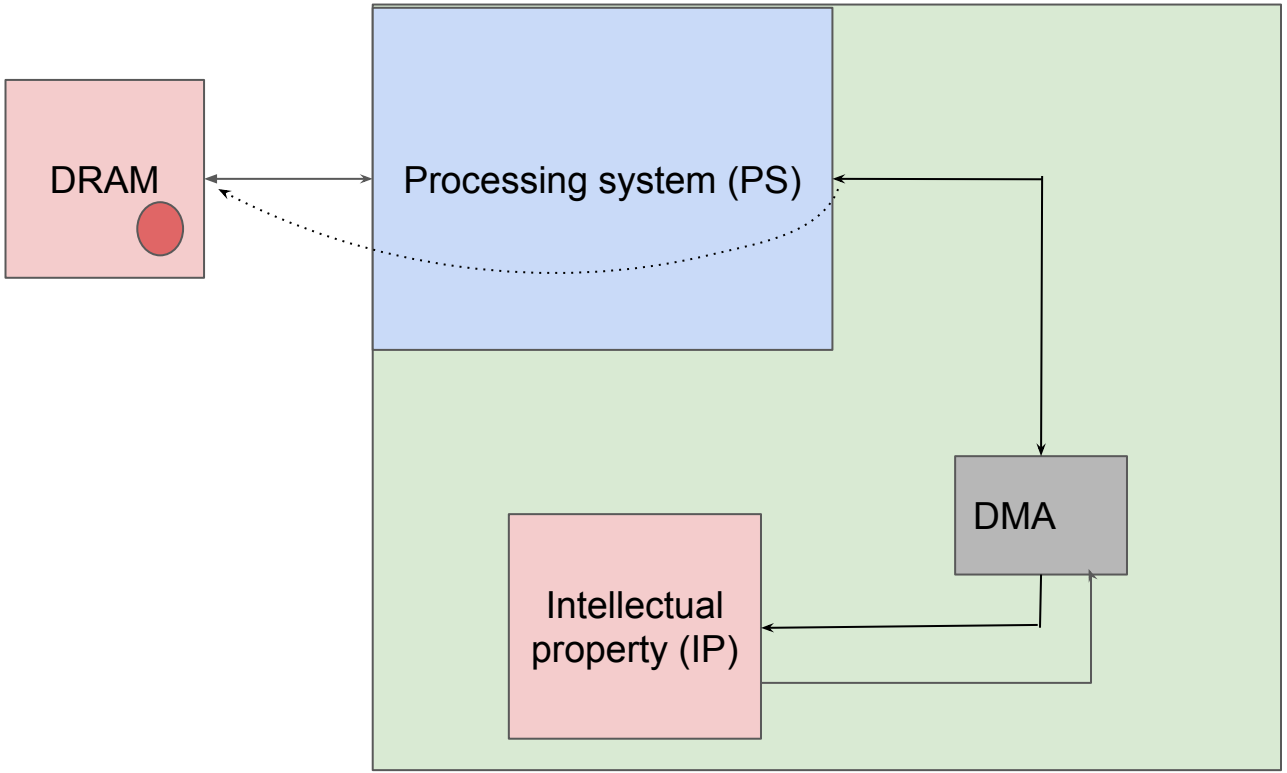
When IP is fast enough to send the data to DMA, but DMA not aware about it.

*DMA- Direct memory access

SDK

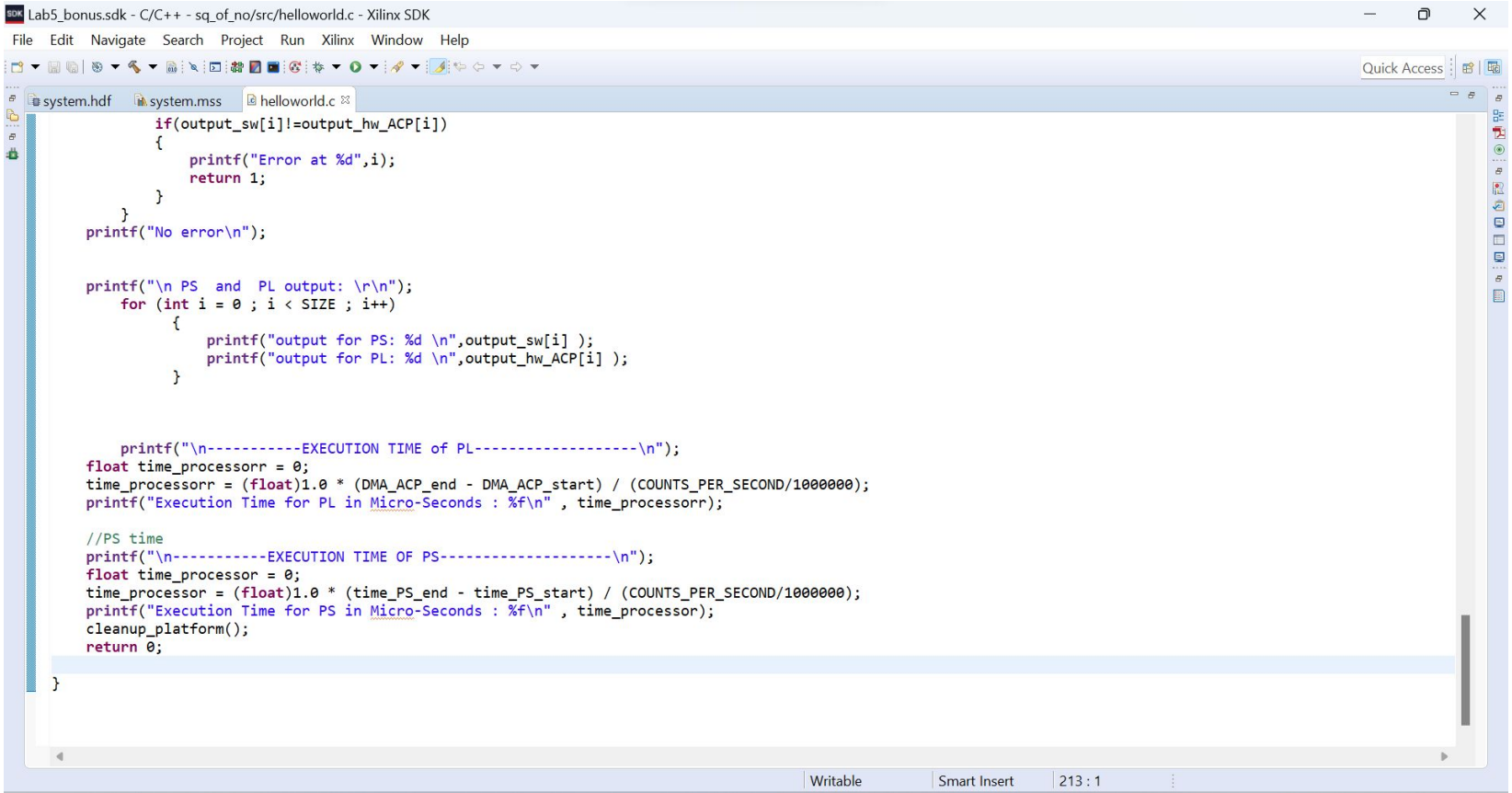

Lab5_bonus.sdk - C/C++ - sq_of_no/src/helloworld.c - Xilinx SDK
File Edit Navigate Search Project Run Xilinx Window Help
Quick Access
system.hdf
system.mss
helloworld.c
{
return 1;//returning of failure
}
// XAxiDMA_readreg(Base add. of dma)
status=XAxiDma_ReadReg(XPAR_AXI_DMA_0_BASEADDR,0x04) & 0x00000002;
// 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_ACP_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

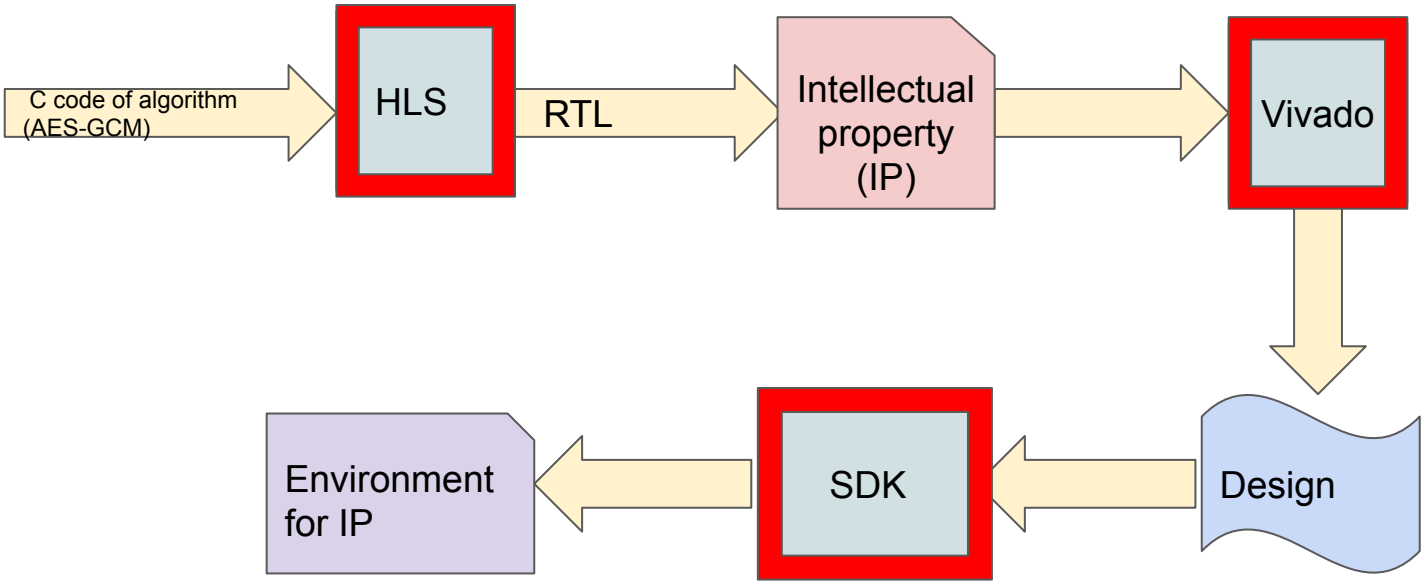
SDK


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

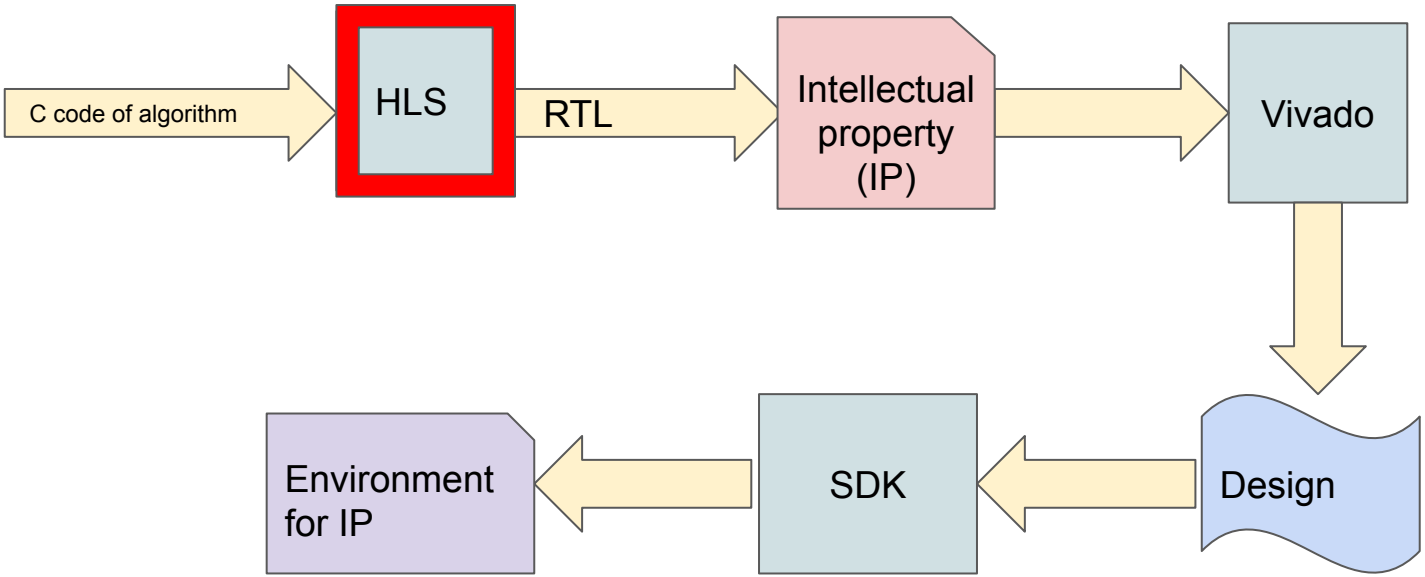
Recap


C code of algorithm
(AES-GCM)
HLS
RTL
Intellectual
property
(IP)
Vivado
Design
SDK
Environment
for IP

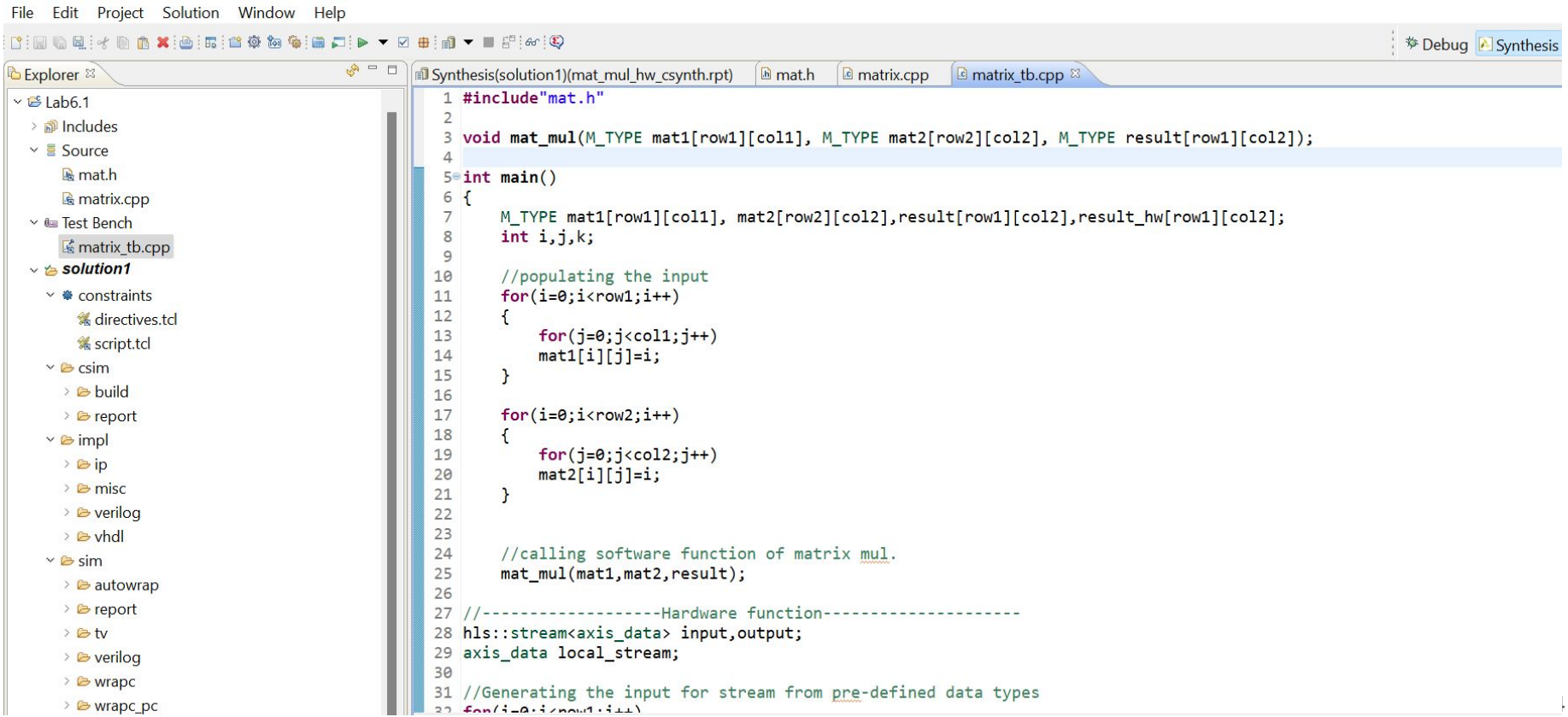
FPGA hands-on

- Matrix multiplication program

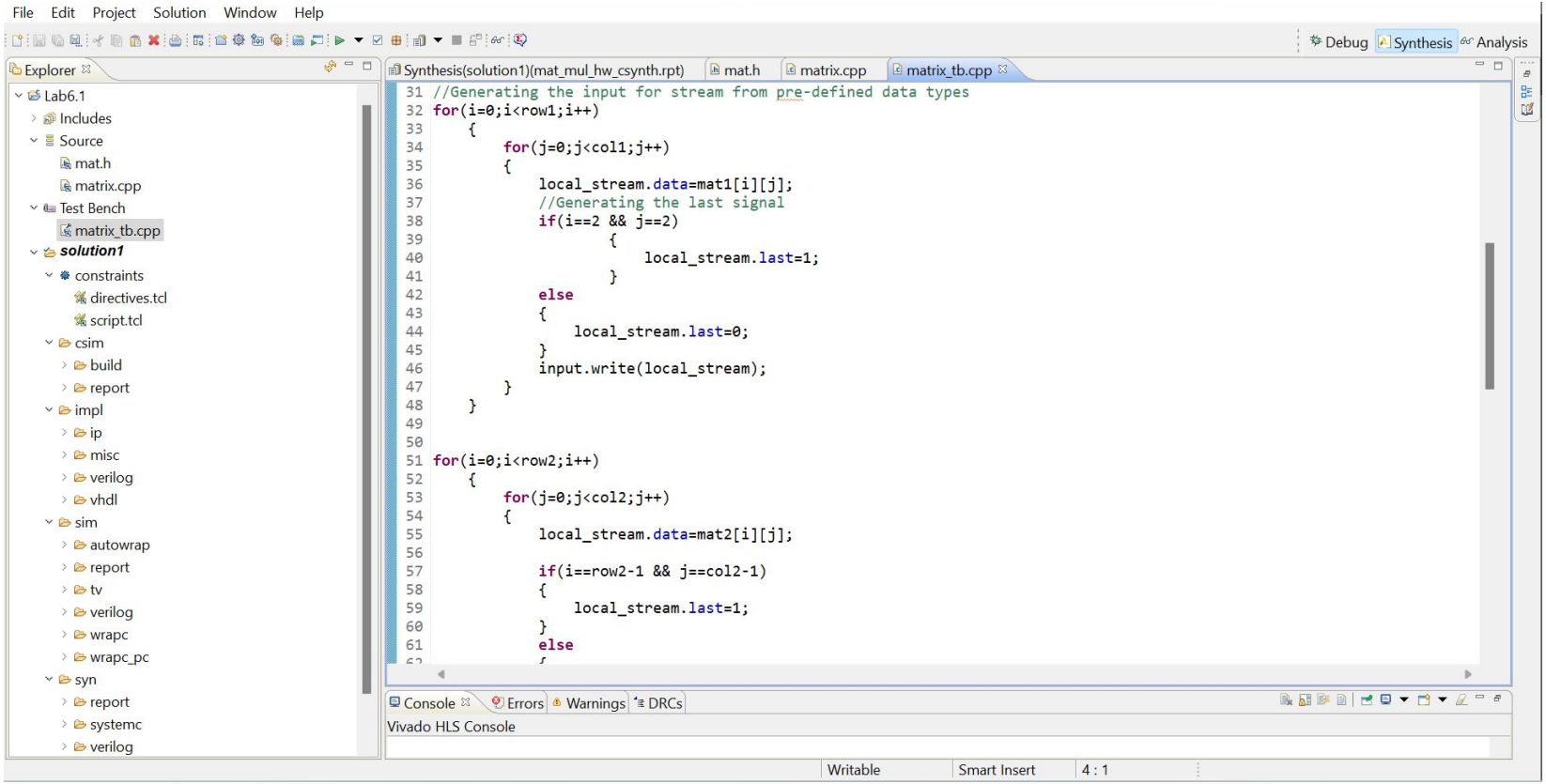
Recap : Workflow



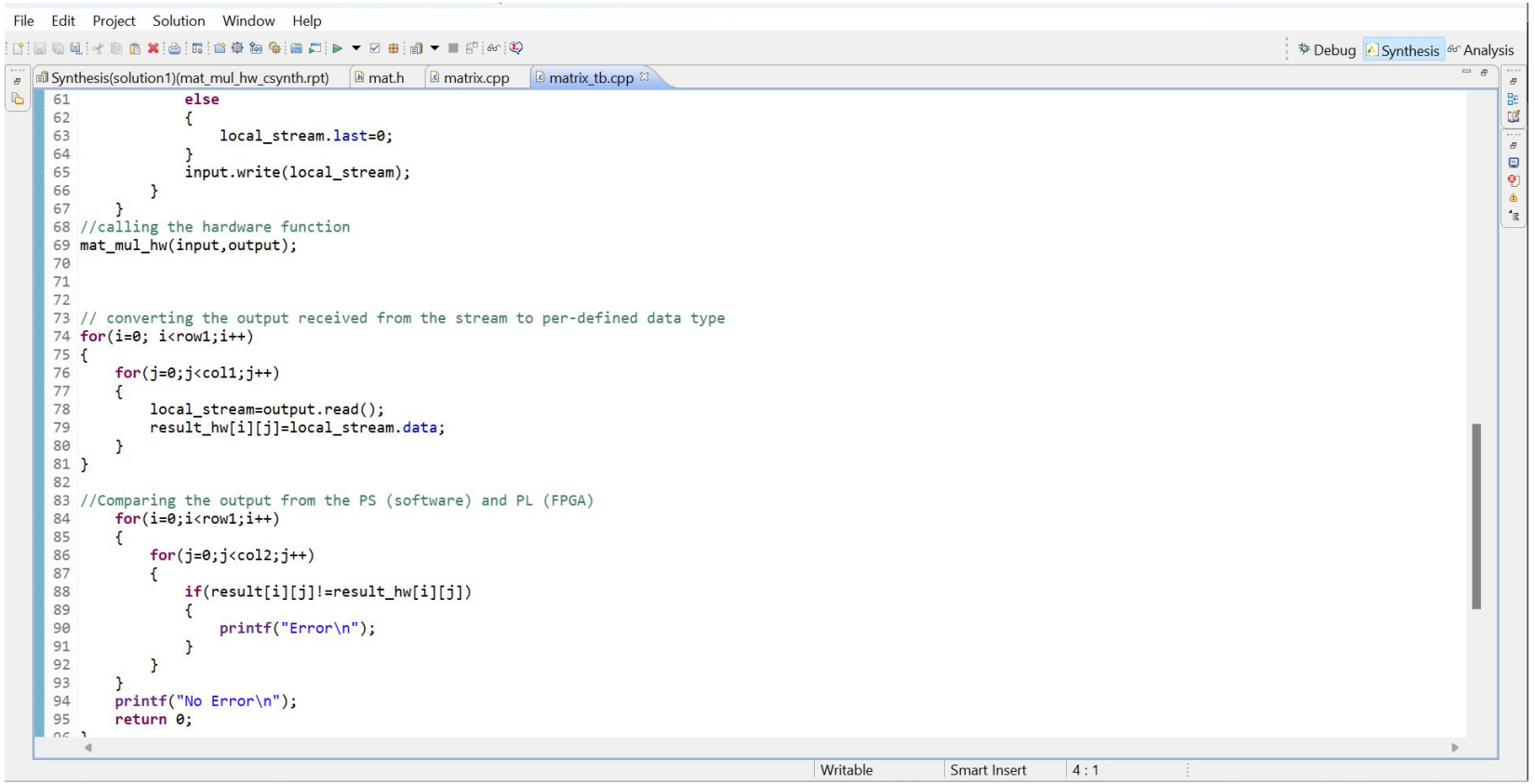
Matrix multiplication program



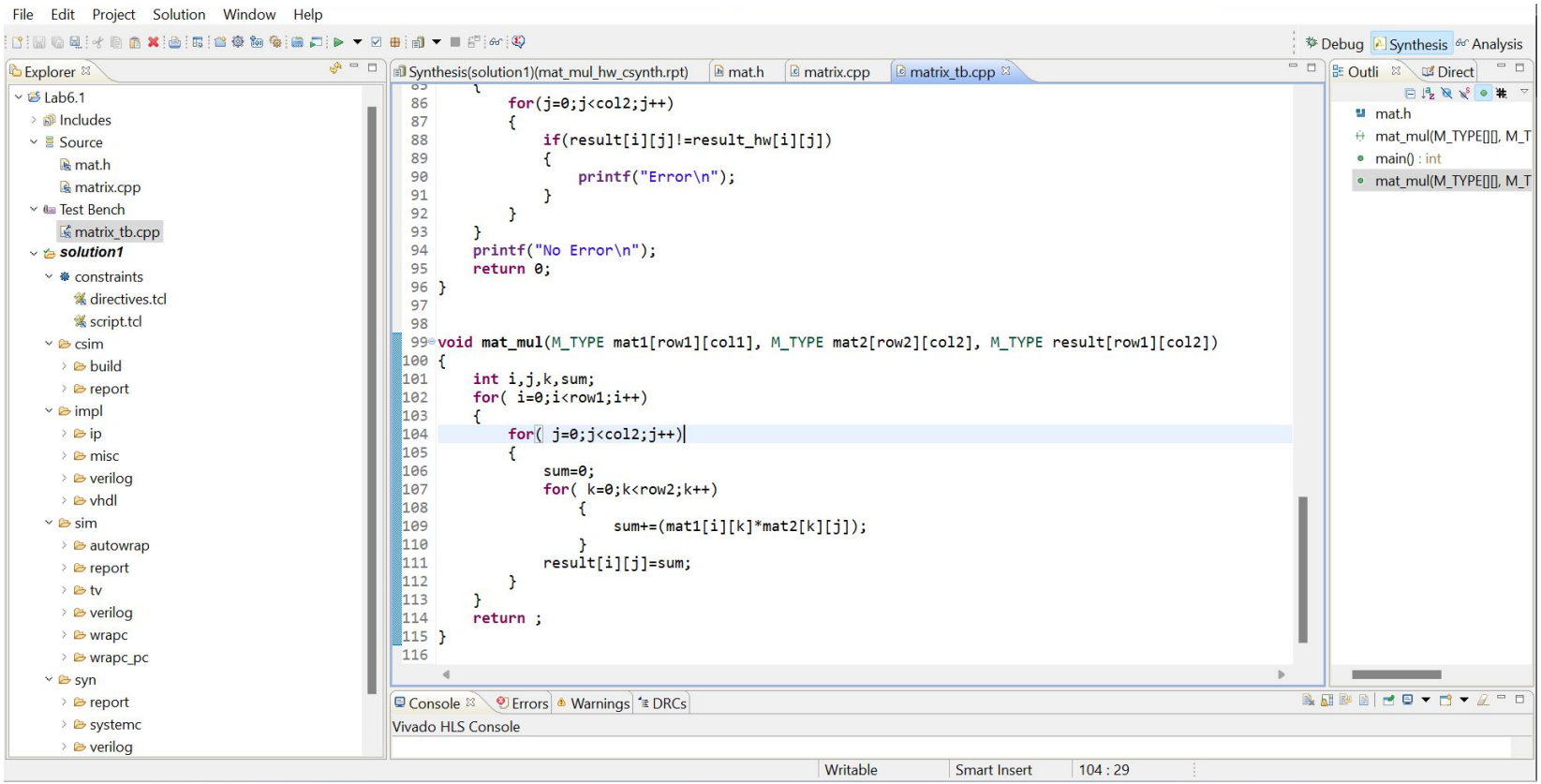
Matrix multiplication program



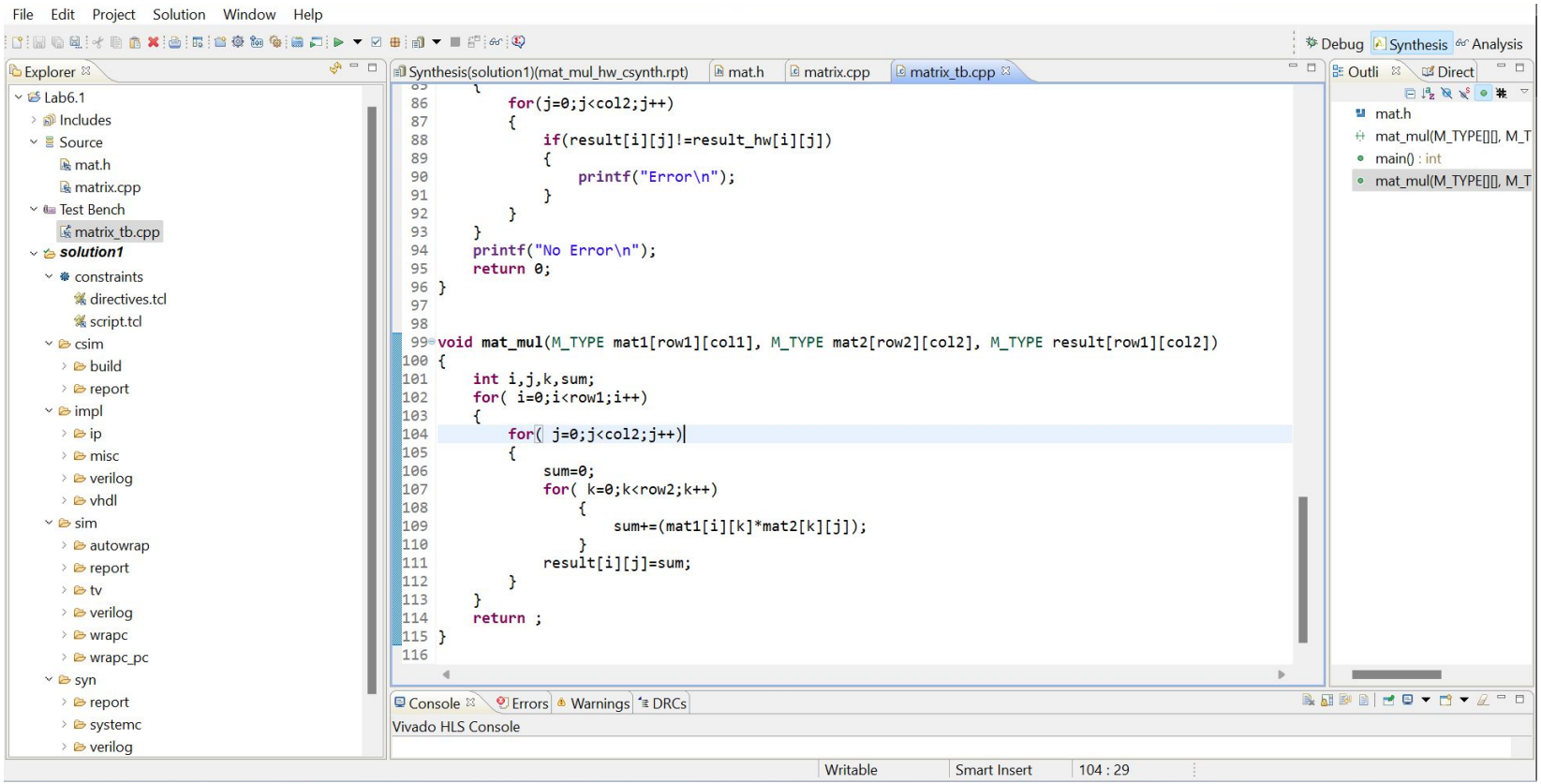
Matrix multiplication program



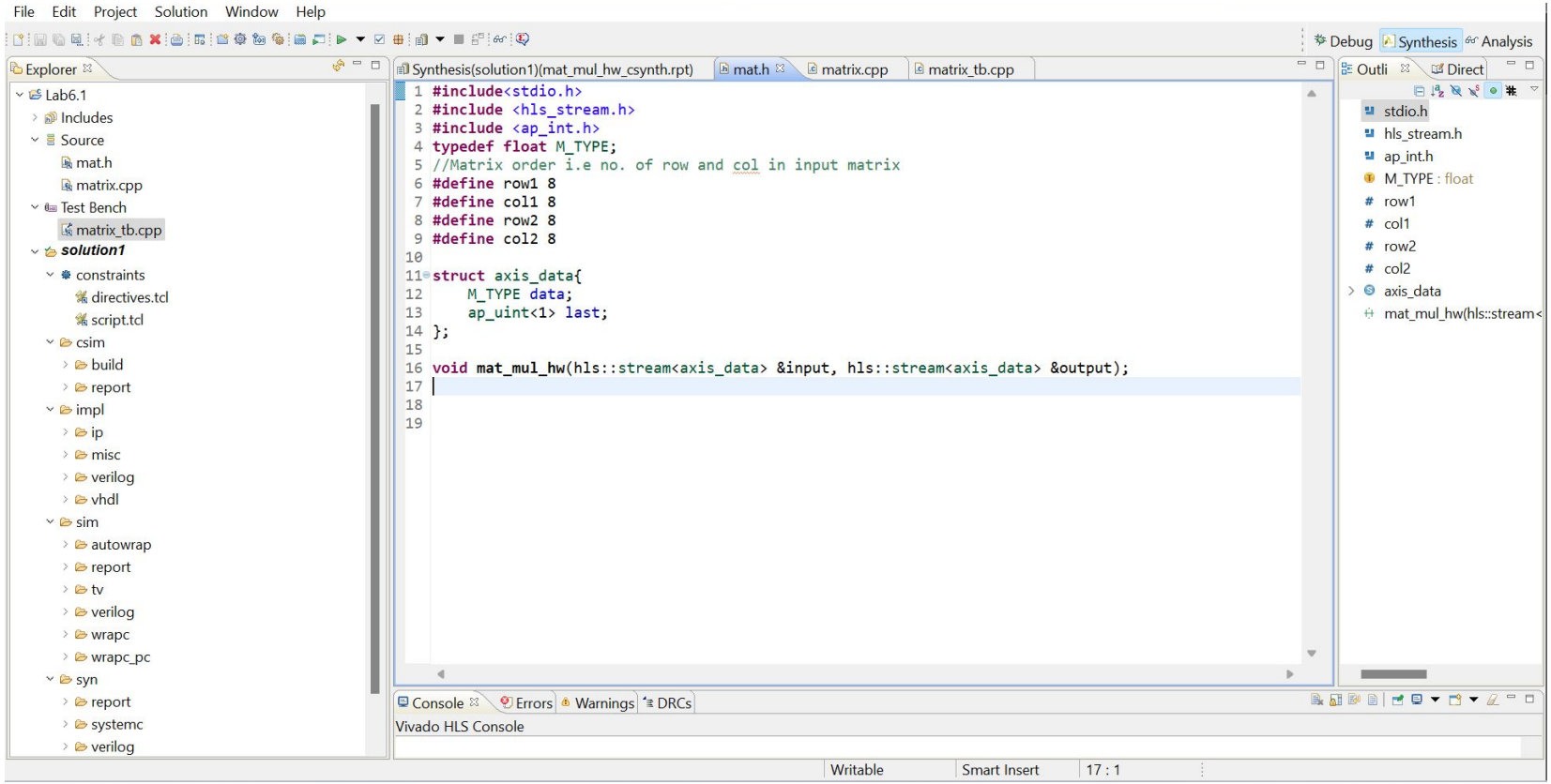
Matrix multiplication program



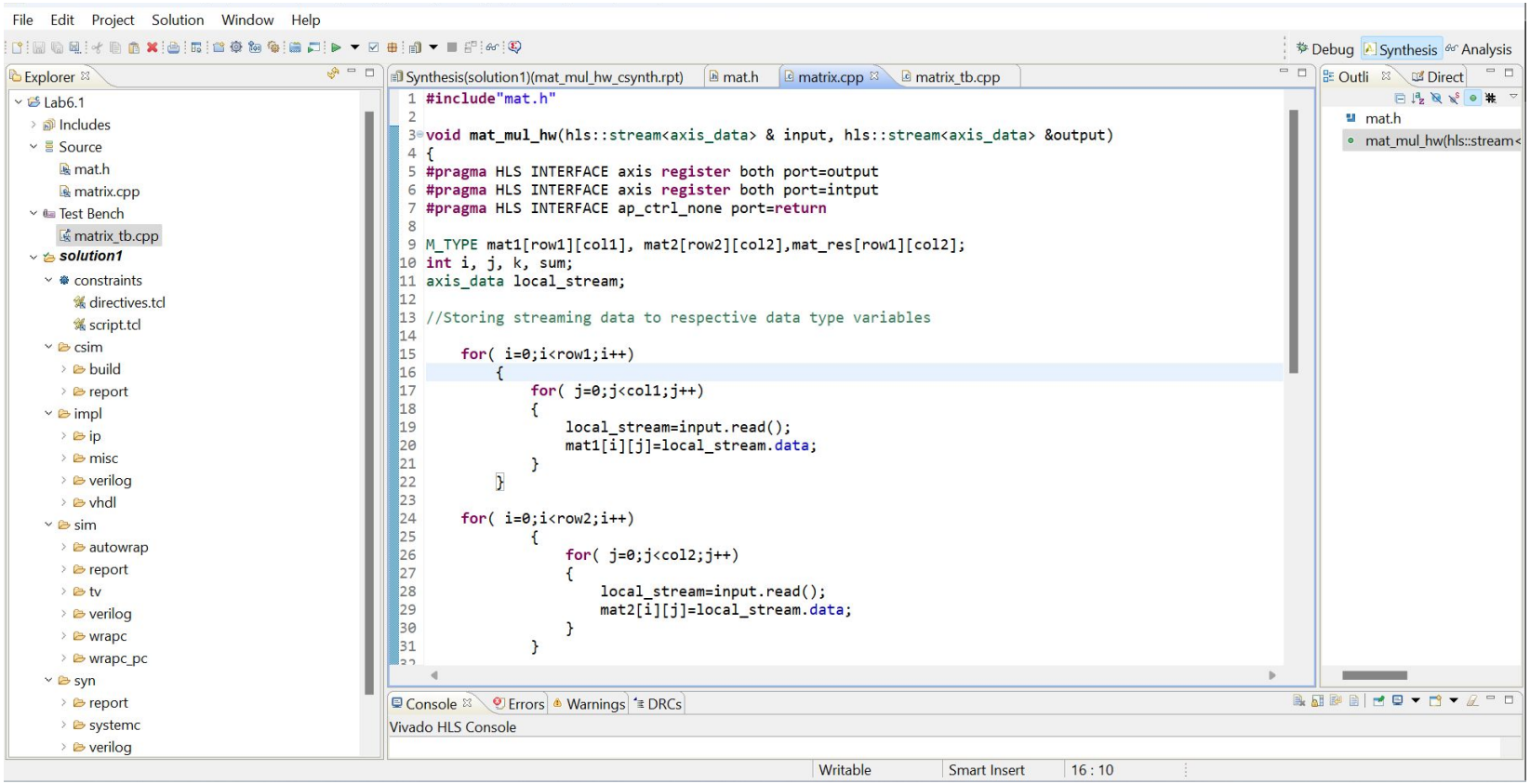
Matrix multiplication program



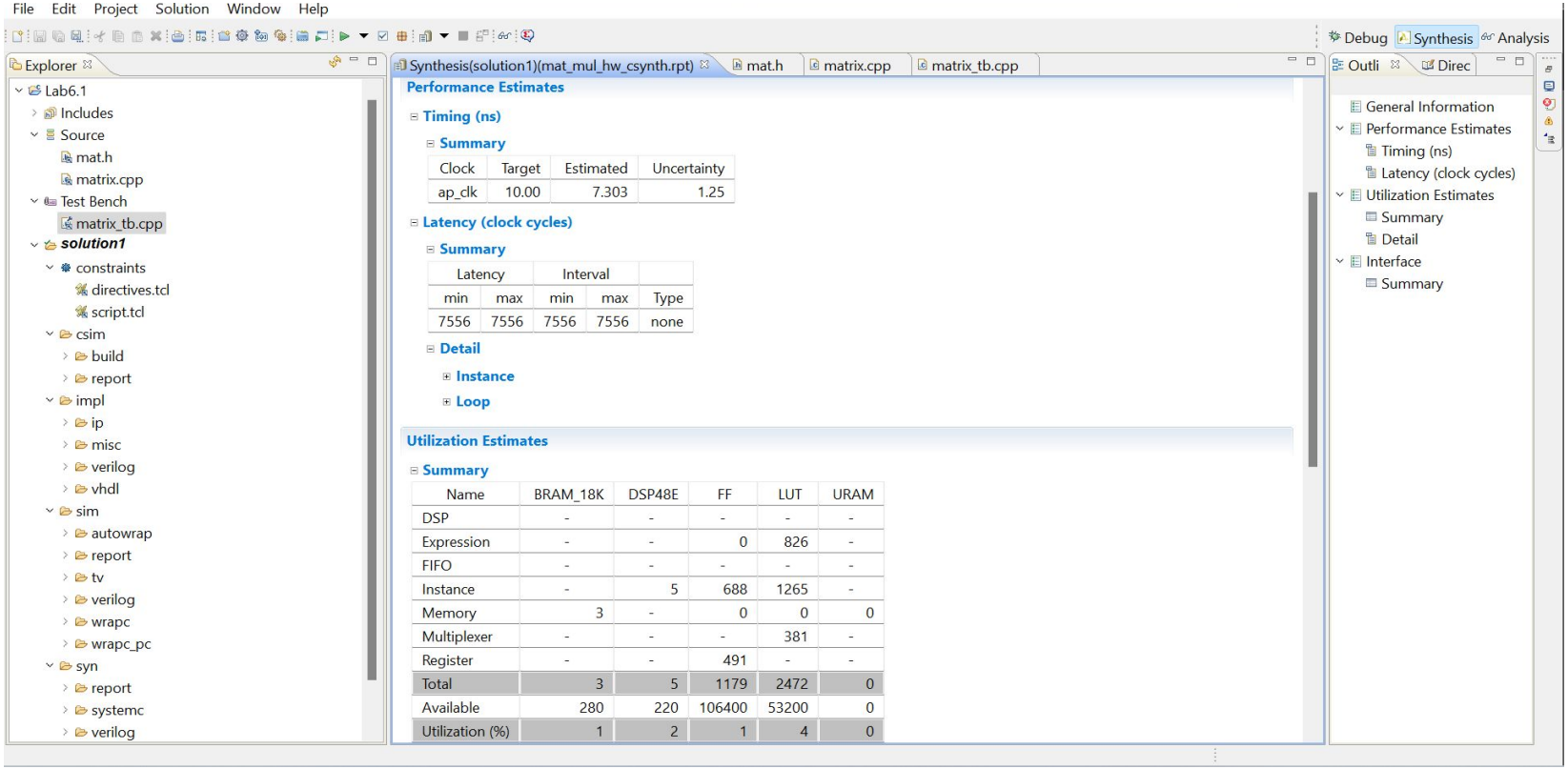
Matrix multiplication program



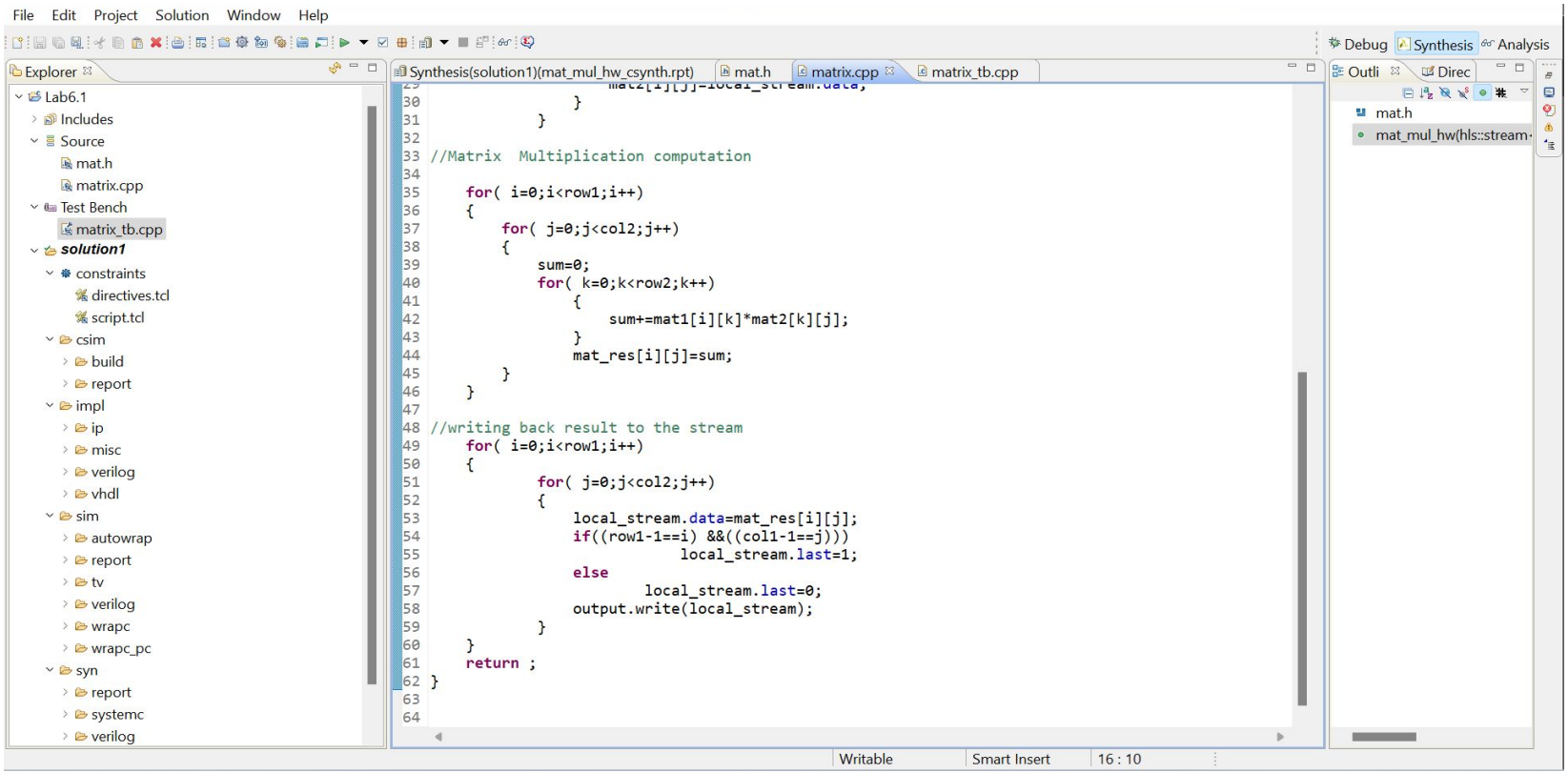
Matrix multiplication program



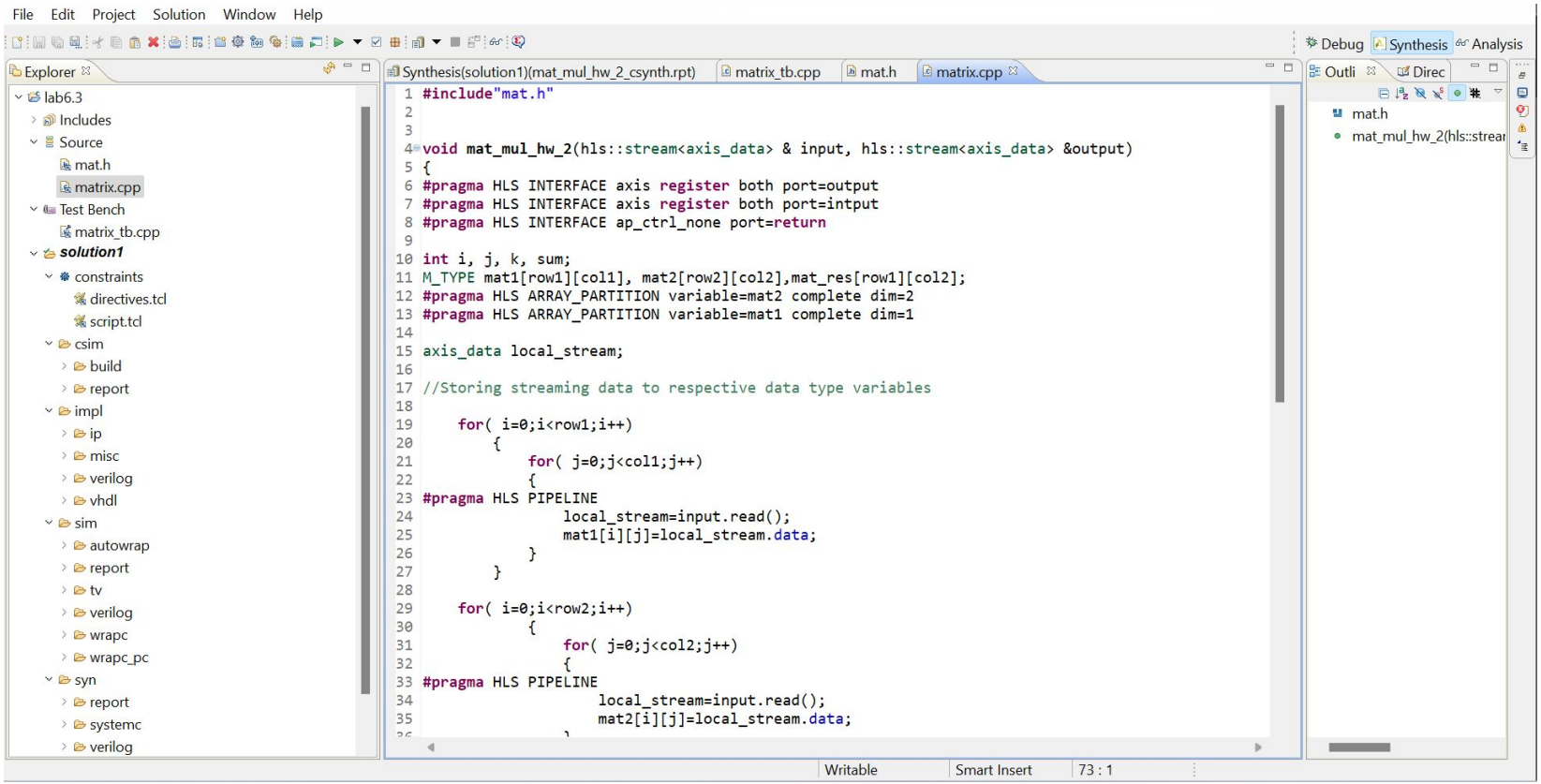
Matrix multiplication program : Analysis



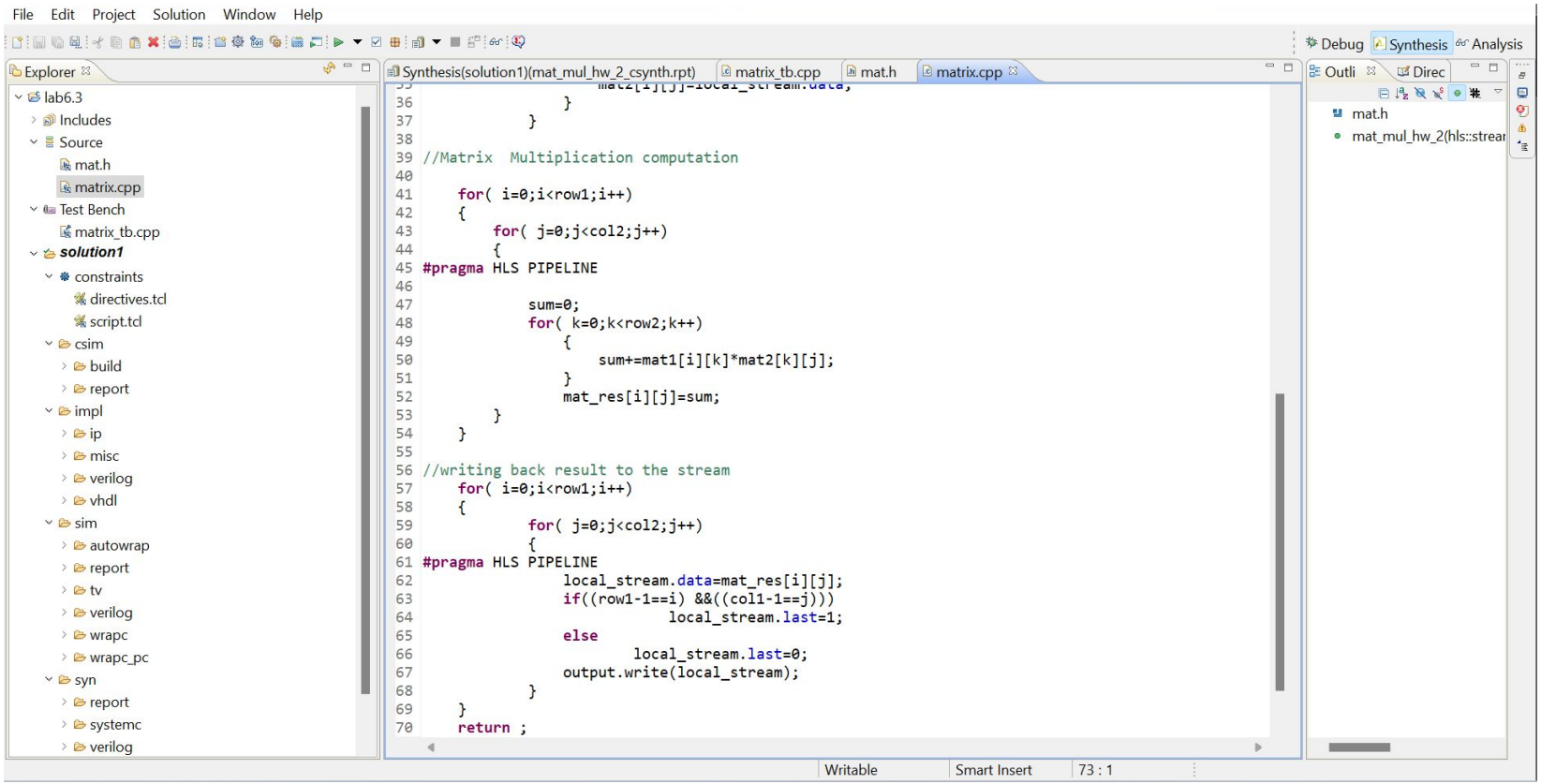
Matrix multiplication program



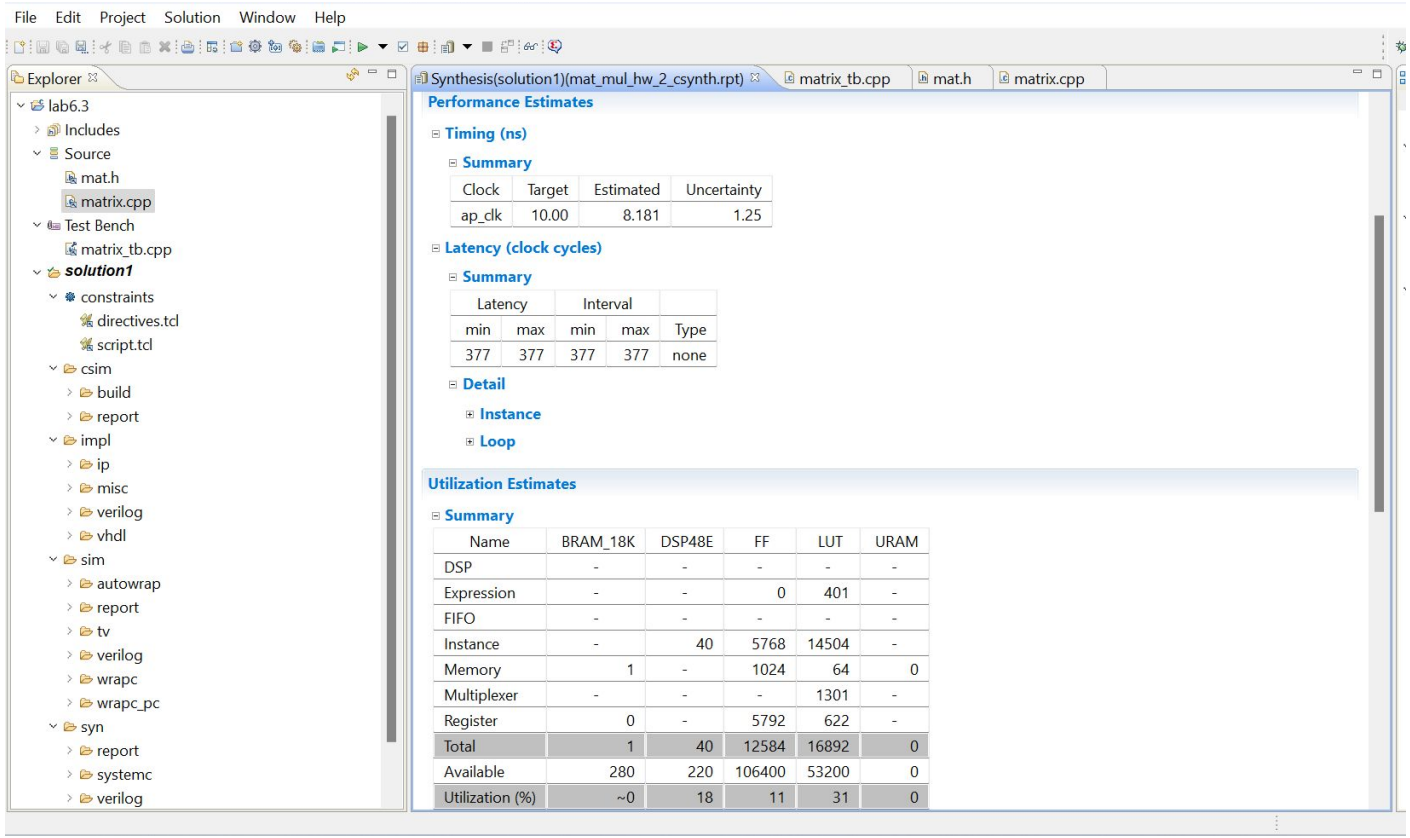
Matrix multiplication program : Optimised



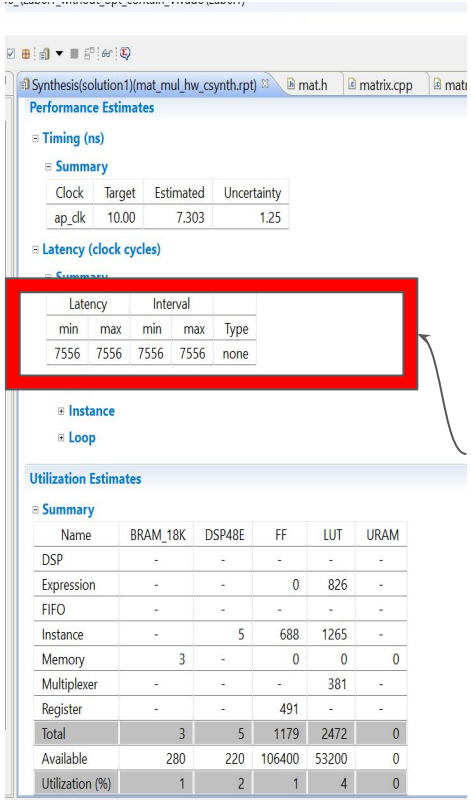
Matrix multiplication program : Optimised



Matrix multiplication Optimised: Analysis



Comparison between optimised and unoptimised



Synthesis(solution1)(mat_mul_hw_csynth.rpt) | mat.h | matrix.cpp

Performance Estimates

Timing (ns)

Summary

Clock	Target	Estimated	Uncertainty
ap_clk	10.00	7.303	1.25

Latency (clock cycles)

Latency		Interval		
min	max	min	max	Type
7556	7556	7556	7556	none

Instance

Loop

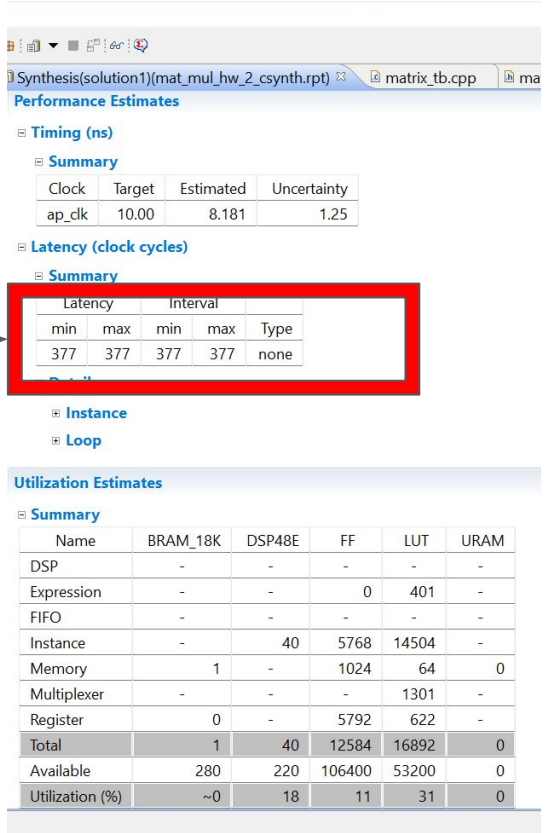
Utilization Estimates

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

optimised

unoptimised



Synthesis(solution1)(mat_mul_hw_2_csynth.rpt) | matrix_tb.cpp | ma

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

Instance

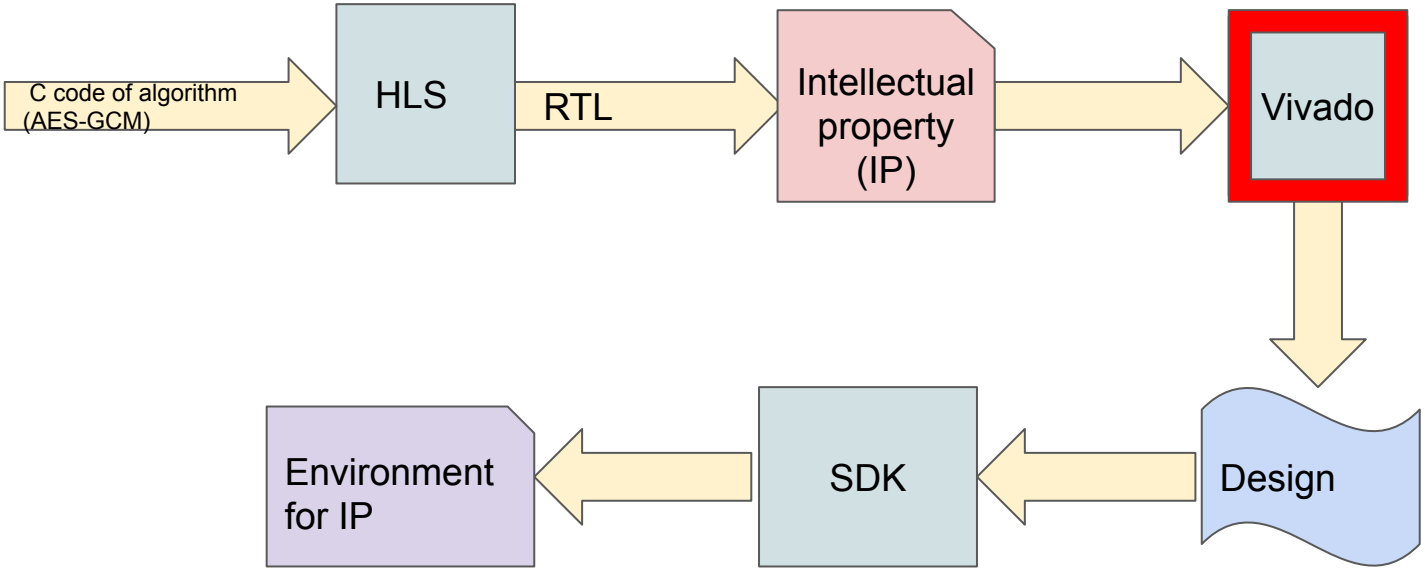
Loop

Utilization Estimates

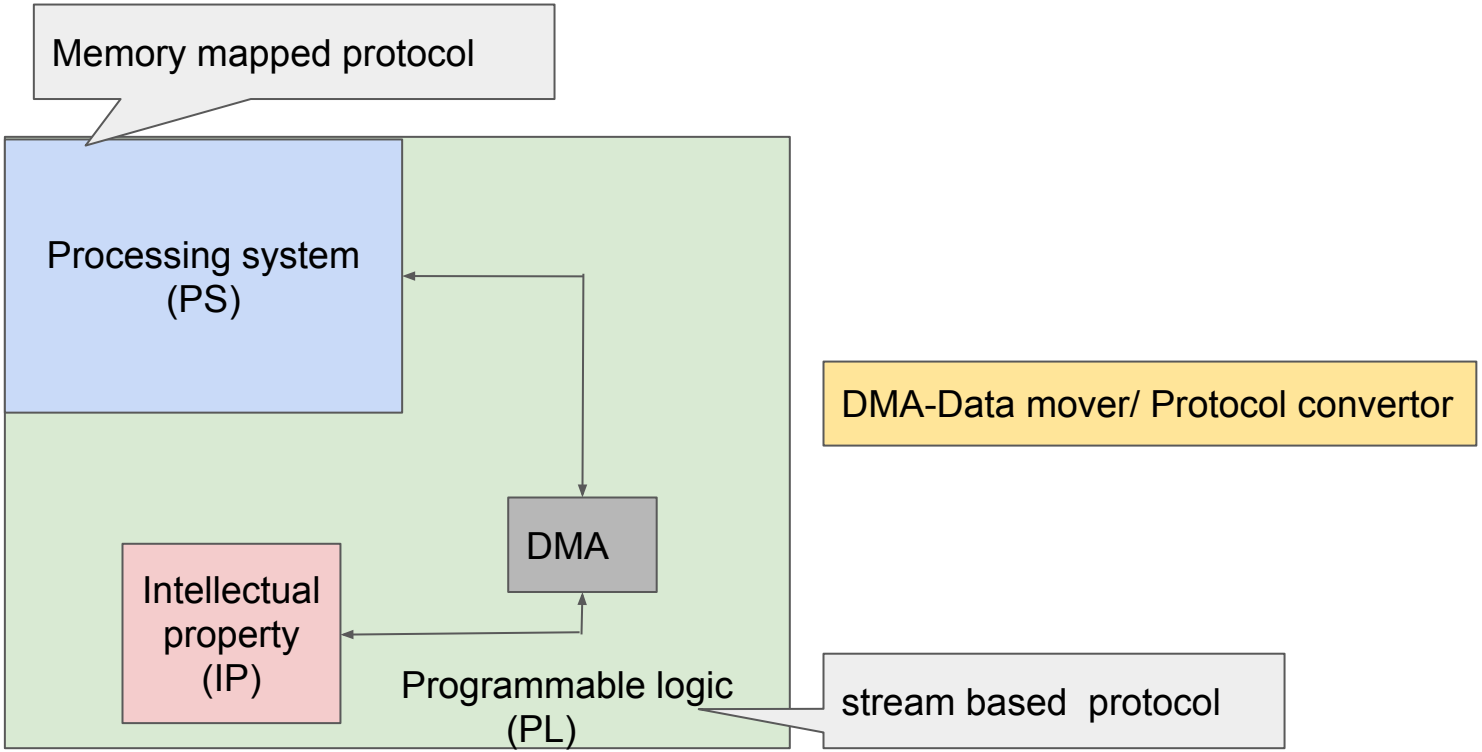
Summary

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

FPGA: Workflow

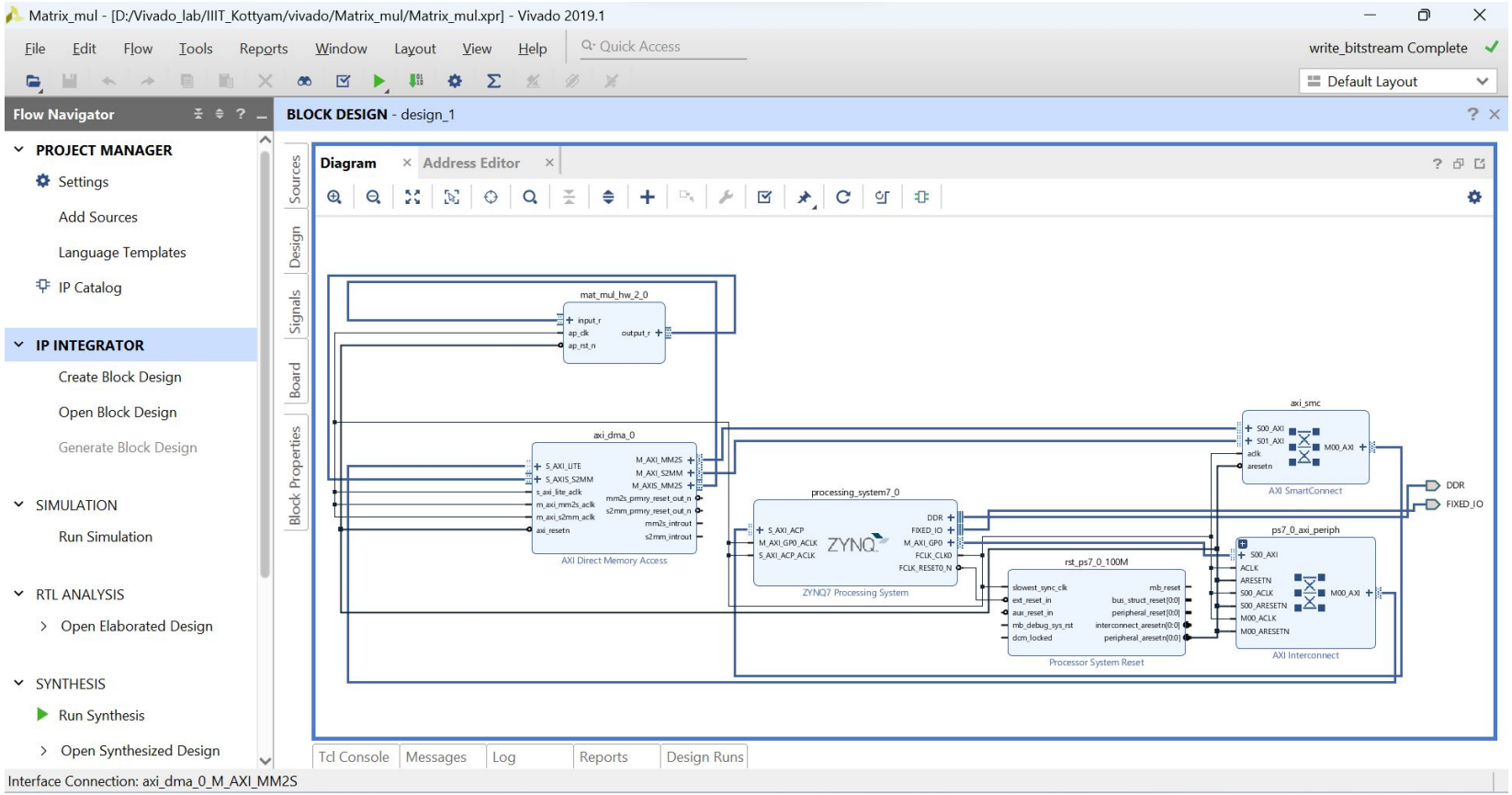


FPGA end-to-end working

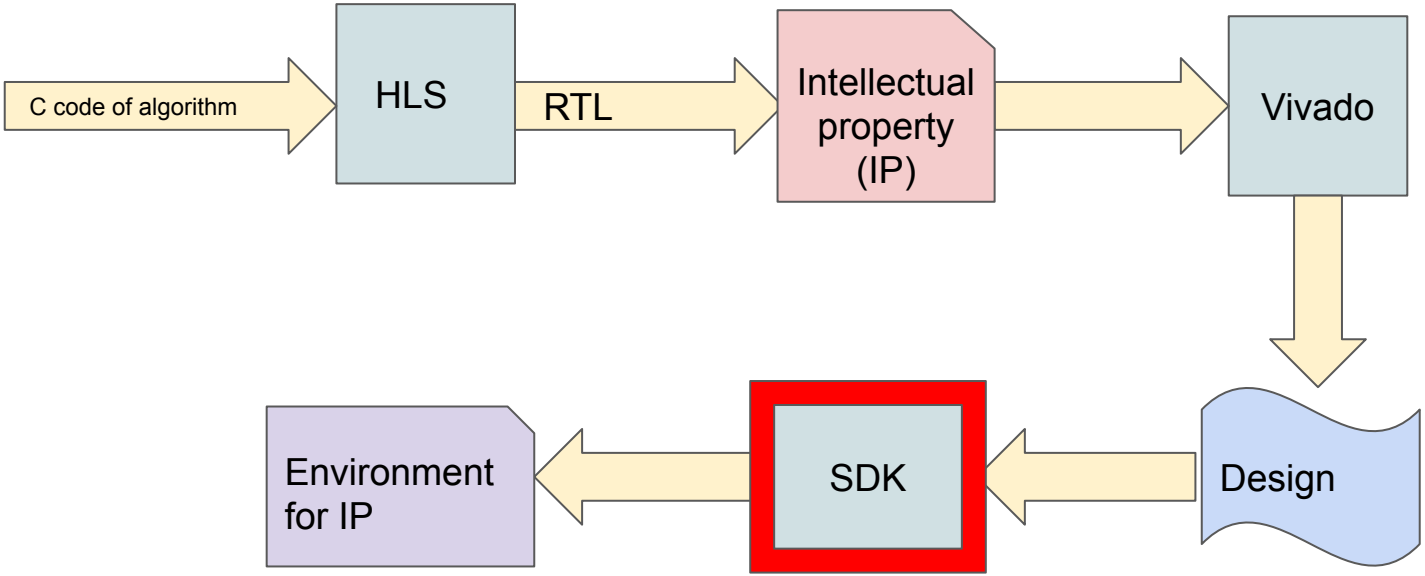


*DMA- Direct memory access

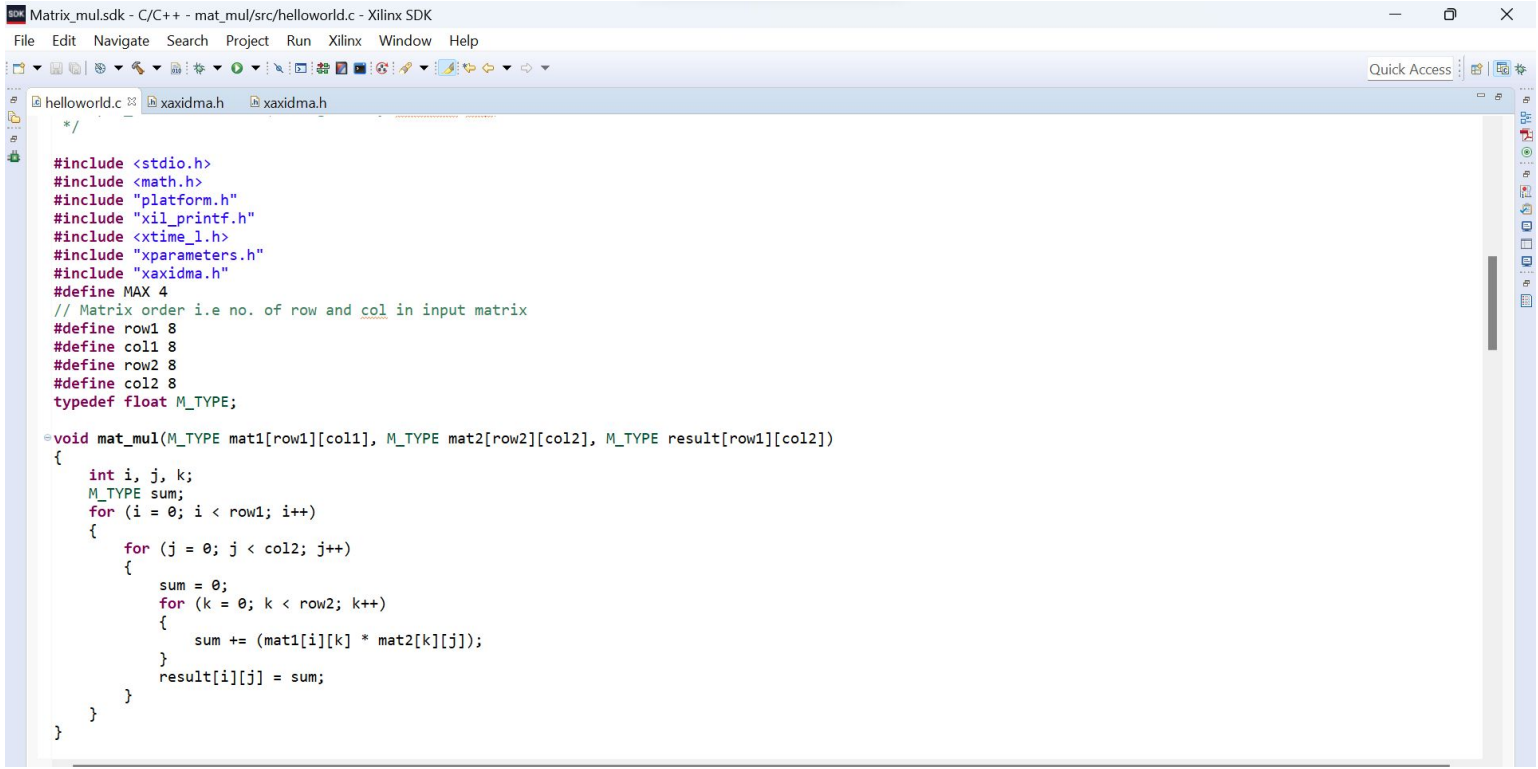
Block diagram: Vivado



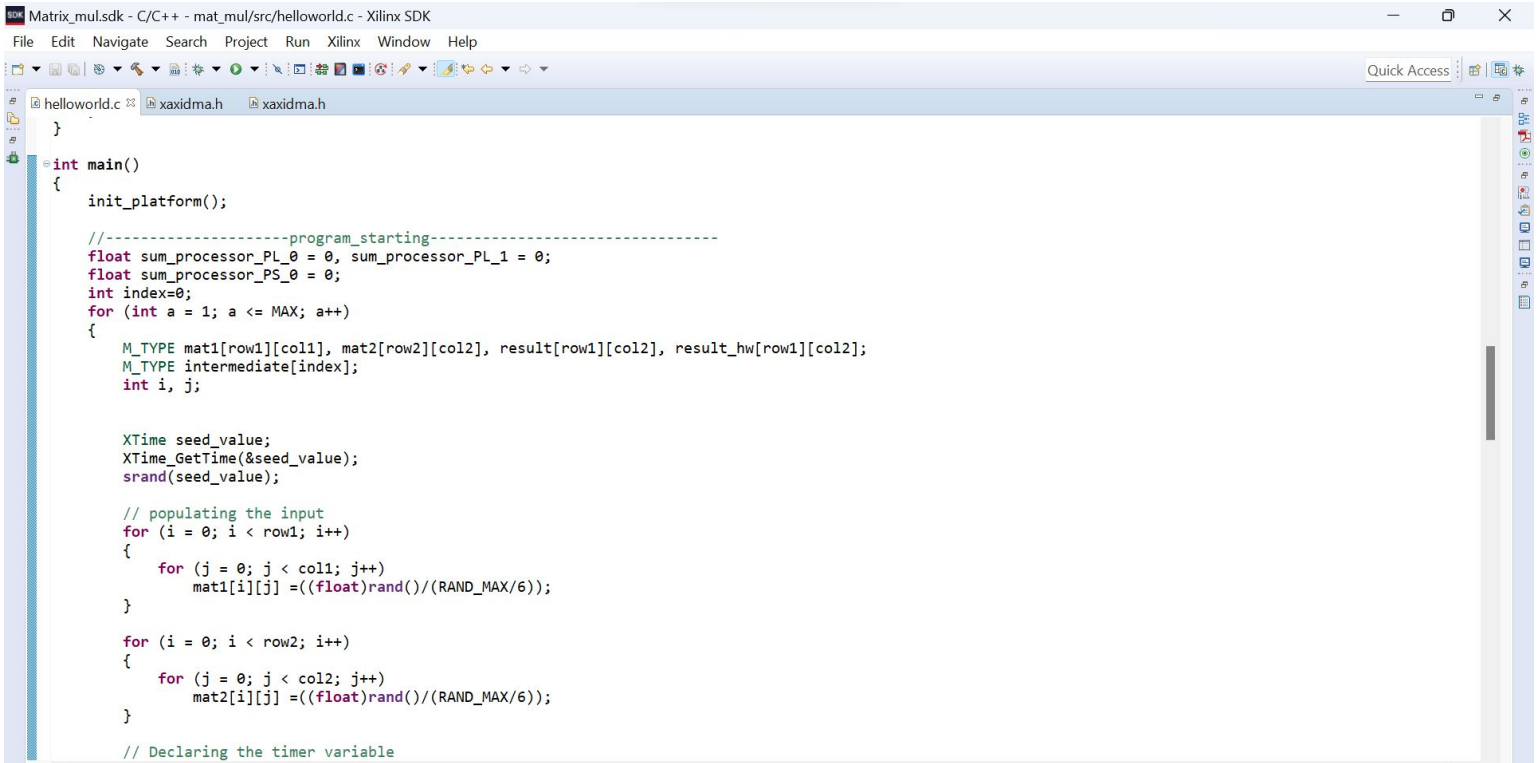
FPGA: Workflow



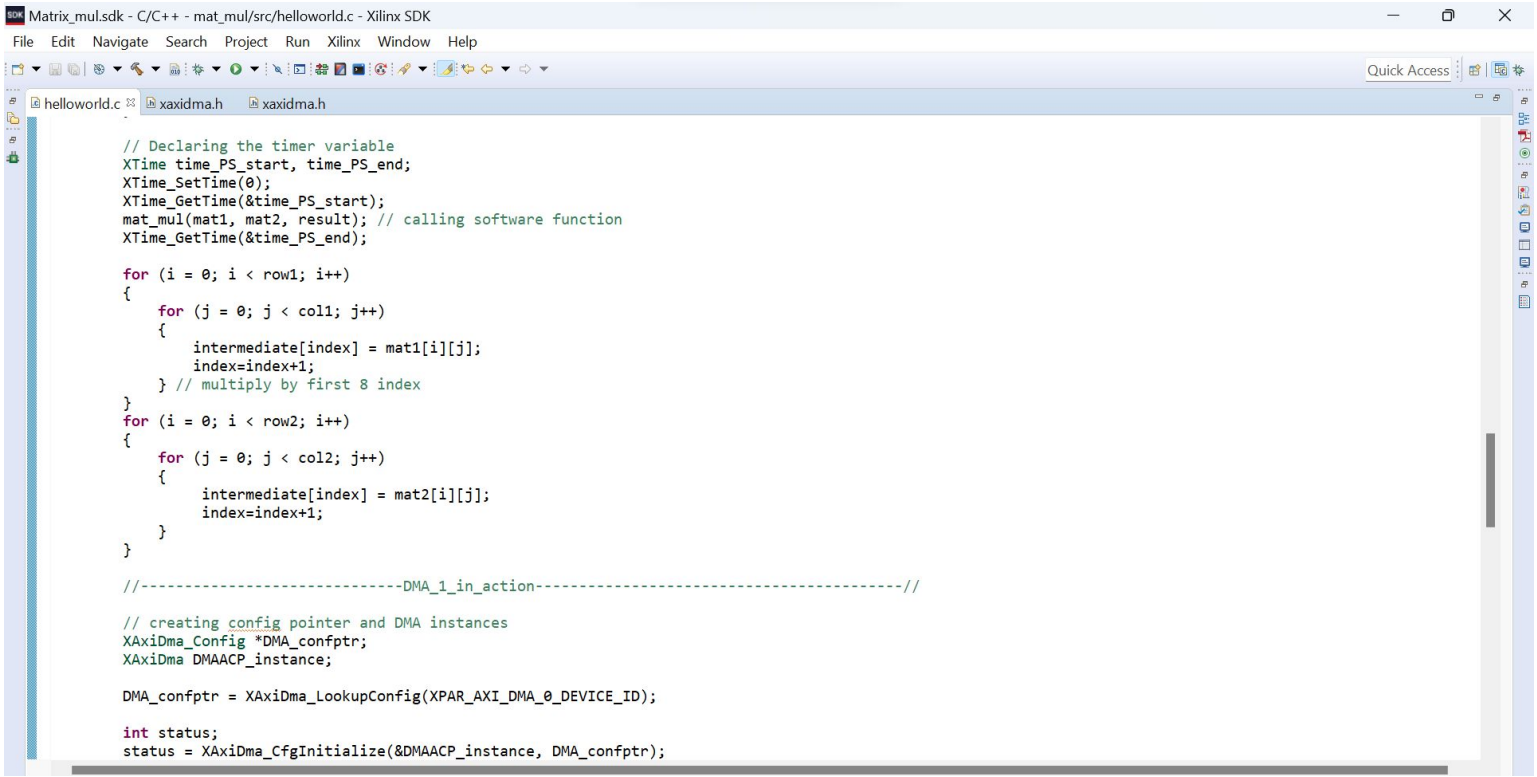
SDK


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
xaxidma.h
xaxidma.h
*/
#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

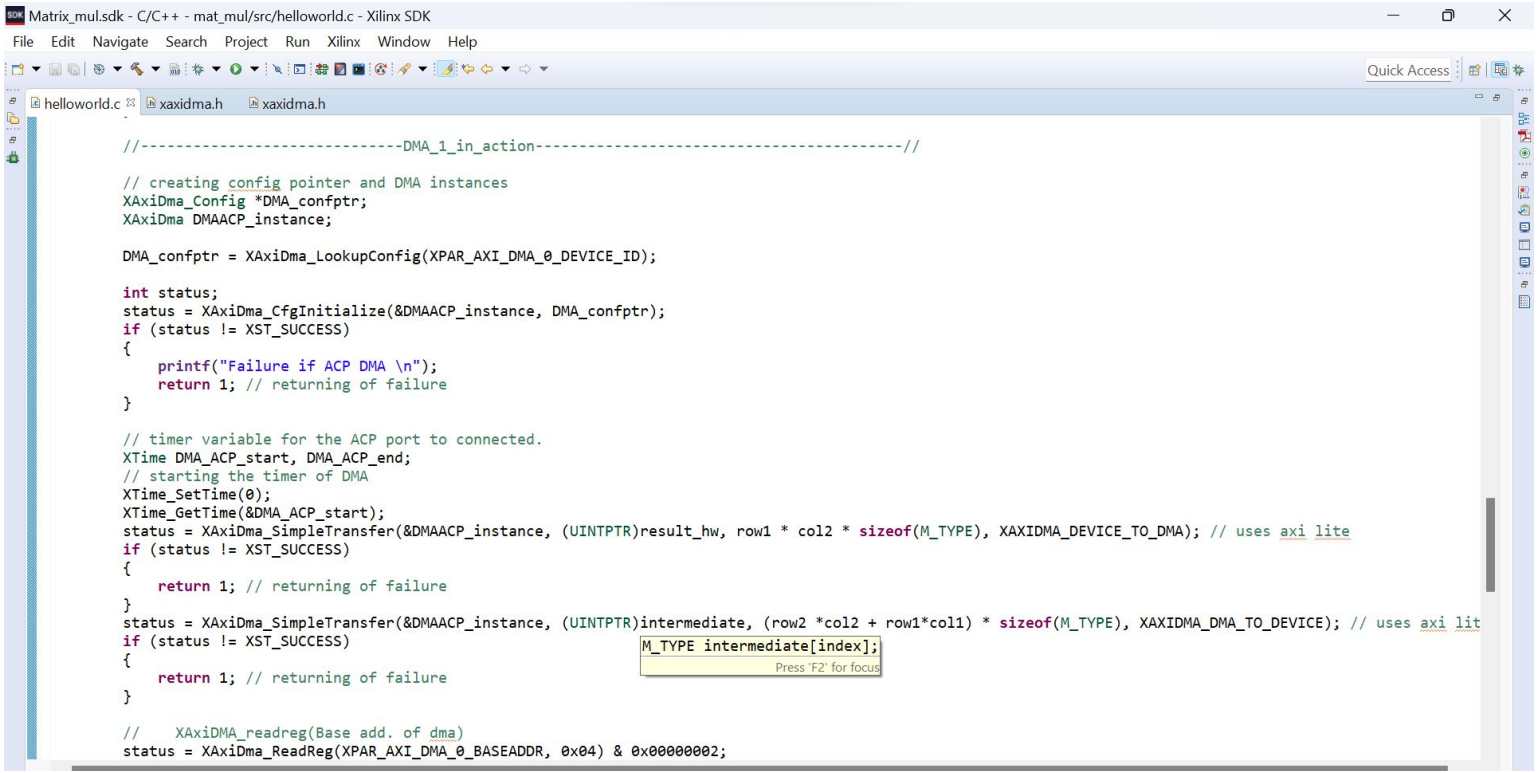

Matrix_mul.sdk - C/C++ - mat_mul/src/helloworld.c - Xilinx SDK
File Edit Navigate Search Project Run Xilinx Window Help
Quick Access
helloworld.c
xaxidma.h
xaxidma.h
}
int main()
{
init_platform();
//--------------------program_starting--------------------------------
float sum_processor_PL_0 = 0, sum_processor_PL_1 = 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

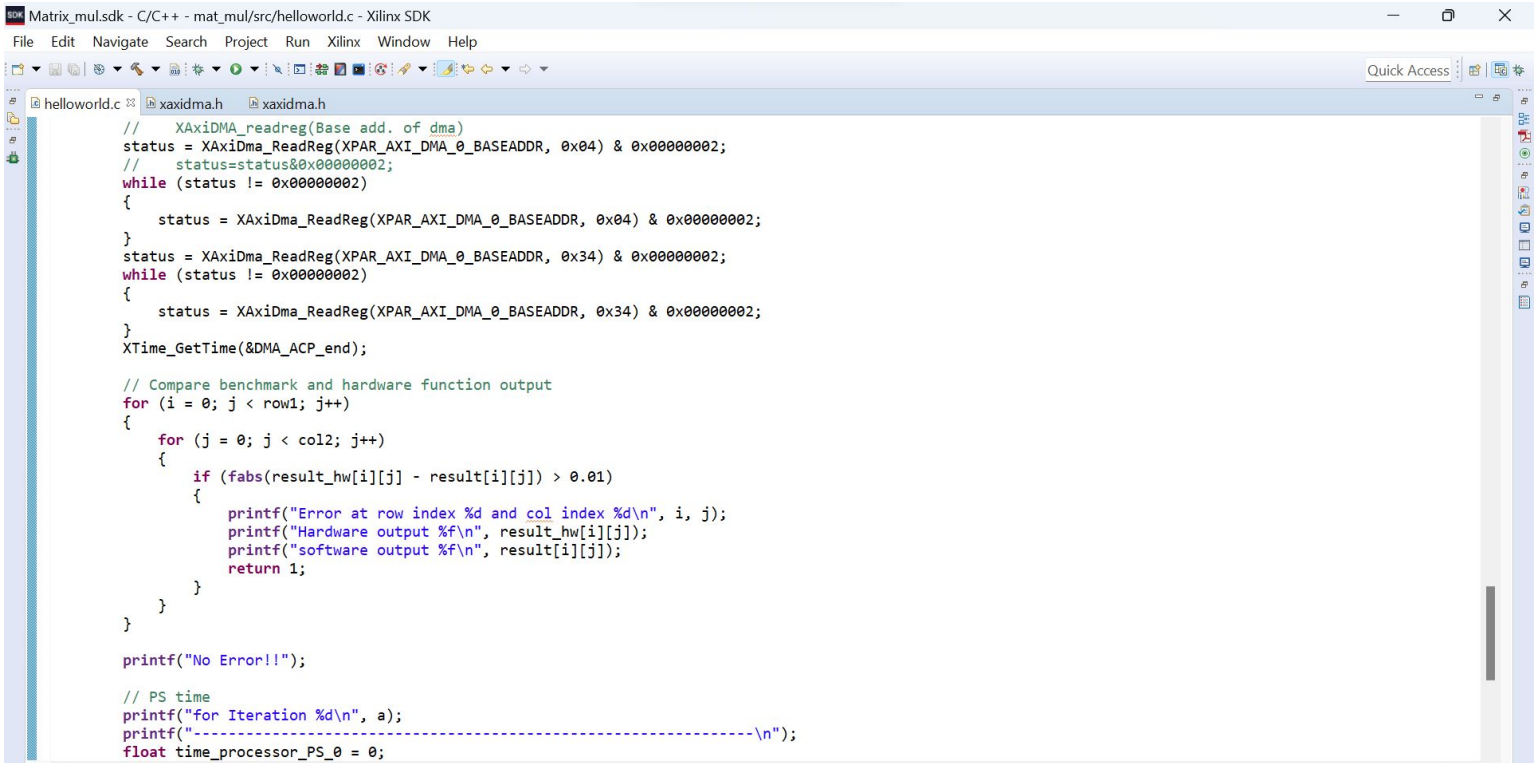



```
// Declaring the timer variable
XTime time_PS_start, time_PS_end;
XTime_SetTime(0);
XTime_GetTime(&time_PS_start);
mat_mul(mat1, mat2, result); // calling software function
XTime_GetTime(&time_PS_end);

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
helloworld.c
xaxidma.h
xaxidma.h
//------------------------------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)
{
return 1; // returning of failure
}
M_TYPE intermediate[index];
Press 'F2' for focus
// XAxiDMA_readreg(Base add. of dma)
status = XAxiDma_ReadReg(XPAR_AXI_DMA_0_BASEADDR, 0x04) & 0x00000002;

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
xaxidma.h
xaxidma.h
//   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 form 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~~