Traineeships in Advanced Computing for High Energy Physics (TAC-HEP)

FPGA module training

Week-6

Lecture-12: 06/03/2025

Varun Sharma

University of Wisconsin – Madison, USA

WISCONSIN
UNIVERSITY OF WISCONSIN-MADISON

Content

- Vivado/Vitis HLS Setup
  - Data types
  - HLS Pragmas

TAC-HEP 2025

Data Types

Data Types

- Data types used in a C/C++ function impact the accuracy of the result and the memory requirements, and can impact the performance

- A 32-bit integer *int* data type can hold more data and therefore provide more precision than an 8-bit char type, but it requires more storage.
- Similarly, when the C/C++ function is to be synthesized to an RTL implementation, the types impact the precision, the area, and the performance of the RTL design

- HLS supports the synthesis of all standard C/C++ types, including exact-width integer types

  `(unsigned) char`, `(unsigned) short`, `(unsigned) int`

  `(unsigned) long`, `(unsigned) long long`

  `(unsigned) intN_t` (where `N` is 8, 16, 32, and 64, as defined in `stdint.h`)

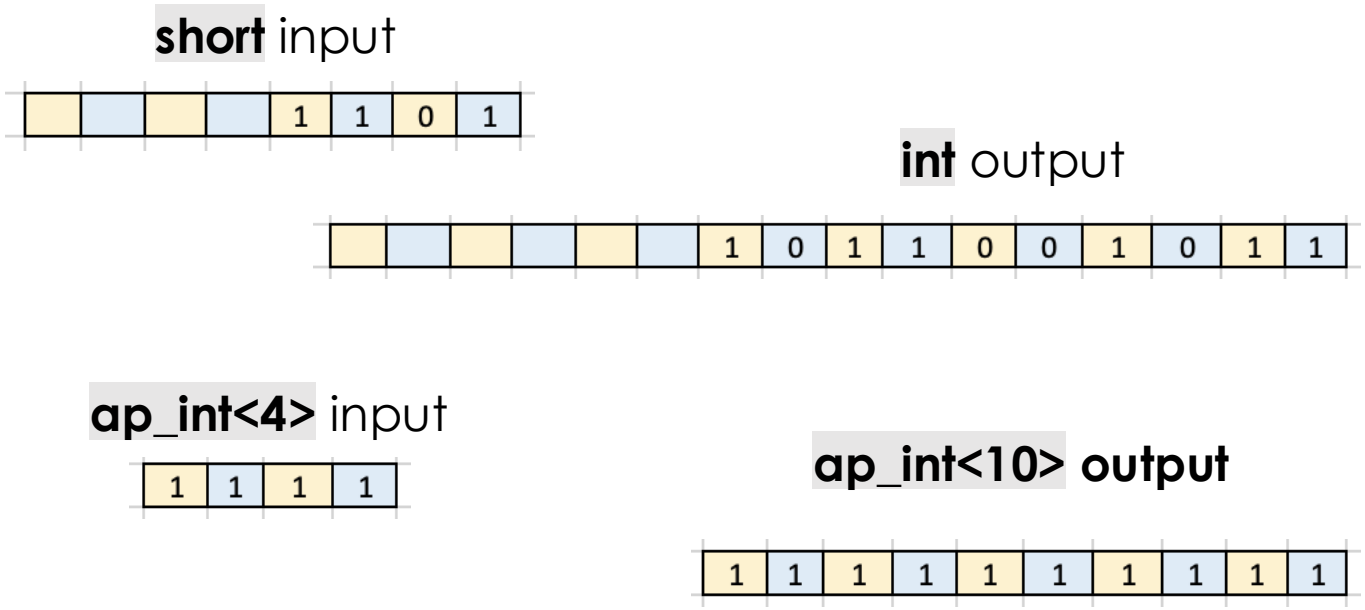
  `float`, `double`

- **Recommended to define the data types for all variables in a common header file, which can be included in all source file**

Arbitrary precision

Creating hardware, it is useful to use more accurate bit-widths

For ex: a case in which the input to a filter is 4-bit and the yielded results requires a maximum of 10-bits



C/C++ data types	Bit-width
(unsigned) char	4
(unsigned) short	8
(unsigned) int	16
(unsigned) long	32
(unsigned) long long	64
float	32
double	64
IntN_t	N=8/16/32/64

Arbitrary precision

Using standard C data types for hardware design results in unnecessary hardware costs.

Operations can use more LUTs and registers than needed for the required accuracy, and delays might even exceed the clock cycle, requiring more cycles to compute the result

C/C++ data types	Bit-width
(unsigned) char	4
(unsigned) short	8
(unsigned) int	16
(unsigned) long	32
(unsigned) long long	64
float	32
double	64
IntN_t	N=8/16/32/64

Simple arithmetic example

```
void basic_arith(
 dinA_t  inA,
 dinB_t  inB,
 dinC_t  inC,
 dinD_t  inD,
 dout1_t *out1,
 dout2_t *out2,
 dout3_t *out3,
 dout4_t *out4 ){

 // Basic arithmetic & math.h sqrtf()
 *out1 = inA * inB;
 *out2 = inB + inA;
 *out3 = inC / inA;
 *out4 = inD % inA;
}
```

```
typedef char      dinA_t;
typedef short     dinB_t;
typedef int       dinC_t;
typedef long long dinD_t;

typedef int          dout1_t;
typedef unsigned int dout2_t;
typedef int32_t      dout3_t;
typedef int64_t      dout4_t;

void basic_arith(
     dinA_t inA,
     dinB_t inB,
     dinC_t inC,
     dinD_t inD,
     dout1_t *out1,
     dout2_t *out2,
     dout3_t *out3,
     dout4_t *out4
    );
```

Data-Type (w/o Arbitrary precision)

```
================================================================
== Vivado HLS Report for 'basic_arith'
================================================================
* Date:           Thu Mar  6 08:24:14 2025

* Version:        2020.1 (Build 2897737 on Wed May 27 20:21:37 MDT 2020)
* Project:        basic_arith_proj
* Solution:       solution1
* Product family: virtexuplus
* Target device:  xcvu9p-flga2104-1-i


================================================================
== Performance Estimates
================================================================
+ Timing:
    * Summary:
    +--------+----------+---------+------------+
    |  Clock |  Target  | Estimated| Uncertainty|
    +--------+----------+---------+------------+
    |ap_clk  | 25.00 ns | 2.846 ns |   3.12 ns  |
    +--------+----------+---------+------------+
```

```
+ Latency:
    * Summary:
    +---------+---------+----------+----------+-----+-----+---------+
    |  Latency (cycles) |  Latency (absolute) |  Interval | Pipeline|
    |   min   |   max   |    min   |    max   | min | max |   Type  |
    +---------+---------+----------+----------+-----+-----+---------+
    |       67|       67| 1.675 us | 1.675 us |   67|   67|   none  |
    +---------+---------+----------+----------+-----+-----+---------+
```

```
================================================================
== Utilization Estimates
================================================================
* Summary:
+---------------------+---------+-------+---------+---------+------+
|         Name        | BRAM_18K| DSP48E|    FF   |   LUT   | URAM|
+---------------------+---------+-------+---------+---------+------+
|DSP                  |        -|      1|        -|        -|     -|
|Expression           |        -|      -|        0|       24|     -|
|FIFO                 |        -|      -|        -|        -|     -|
|Instance             |        -|      -|     1173|      707|     -|
|Memory               |        -|      -|        -|        -|     -|
|Multiplexer          |        -|      -|        -|      309|     -|
|Register             |        -|      -|       68|        -|     -|
+---------------------+---------+-------+---------+---------+------+
|Total                |        0|      1|     1241|     1040|     0|
+---------------------+---------+-------+---------+---------+------+
|Available SLR        |     1440|   2280|   788160|   394080|   320|
+---------------------+---------+-------+---------+---------+------+
|Utilization SLR (%)  |        0|     ~0|       ~0|       ~0|     0|
+---------------------+---------+-------+---------+---------+------+
|Available            |     4320|   6840|  2364480|  1182240|   960|
+---------------------+---------+-------+---------+---------+------+
|Utilization (%)      |        0|     ~0|       ~0|       ~0|     0|
+---------------------+---------+-------+---------+---------+------+
```

Data-Type (w/o Arbitrary precision)

+ Detail:
  * Instance:

Instance	Module	BRAM_18K	DSP48E	FF	LUT	URAM
basic_arith_sdiv_cud_U2	basic_arith_sdiv_cud	0	0	394	238	0
basic_arith_srem_bkb_U1	basic_arith_srem_bkb	0	0	779	469	0
Total		0	0	1173	707	0

  * DSP48E:

Instance	Module	Expression
basic_arith_mul_mdEe_U3	basic_arith_mul_mdEe	i0 * i1

  * Expression:

Variable Name	Operation	DSP48E	FF	LUT	Bitwidth P0	Bitwidth P1
add_ln15_fu_125_p2	+	0	0	24	17	17
Total		0	0	24	17	17

  * Multiplexer:

Name	LUT	Input Size	Bits	Total Bits
ap_NS_fsm	309	69	1	69
Total	309	69	1	69

  * Register:

Name	FF	LUT	Bits	Const Bits
ap_CS_fsm	68	0	68	0
Total	68	0	68	0

== Interface

* Summary:

RTL Ports	Dir	Bits	Protocol	Source Object	C Type
ap_clk	in	1	ap_ctrl_hs	basic_arith	return value
ap_rst	in	1	ap_ctrl_hs	basic_arith	return value
ap_start	in	1	ap_ctrl_hs	basic_arith	return value
ap_done	out	1	ap_ctrl_hs	basic_arith	return value
ap_idle	out	1	ap_ctrl_hs	basic_arith	return value
ap_ready	out	1	ap_ctrl_hs	basic_arith	return value
inA	in	8	ap_none	inA	scalar
inB	in	16	ap_none	inB	scalar
inC	in	32	ap_none	inC	scalar
inD	in	64	ap_none	inD	scalar
out1	out	32	ap_vld	out1	pointer
out1_ap_vld	out	1	ap_vld	out1	pointer
out2	out	32	ap_vld	out2	pointer
out2_ap_vld	out	1	ap_vld	out2	pointer
out3	out	32	ap_vld	out3	pointer
out3_ap_vld	out	1	ap_vld	out3	pointer
out4	out	64	ap_vld	out4	pointer
out4_ap_vld	out	1	ap_vld	out4	pointer

Precise data types

```
void basic_arith_ap(
 dinA_t  inA,
 dinB_t  inB,
 dinC_t  inC,
 dinD_t  inD,
 dout1_t *out1,
 dout2_t *out2,
 dout3_t *out3,
 dout4_t *out4 ){

 // Basic arithmetic & math.h sqrtf()
 *out1 = inA * inB;
 *out2 = inB + inA;
 *out3 = inC / inA;
 *out4 = inD % inA;
}
```

```
typedef ap_int<6>  dinA_t;
typedef ap_int<12> dinB_t;
typedef ap_int<22> dinC_t;
typedef ap_int<33> dinD_t;

typedef ap_int<18> dout1_t;
typedef ap_int<13> dout2_t;
typedef ap_int<22> dout3_t;
typedef ap_int<6>  dout4_t;

void basic_arith_ap(
     dinA_t inA,
     dinB_t inB,
     dinC_t inC,
     dinD_t inD,
     dout1_t *out1,
     dout2_t *out2,
     dout3_t *out3,
     dout4_t *out4
    );
```

Data-Type (w/ Arbitrary precision)

```
================================================================
== Vivado HLS Report for 'basic_arith_ap'
================================================================
* Date:           Thu Mar  6 08:26:29 2025

* Version:        2020.1 (Build 2897737 on Wed May 27 20:21:37 MDT 2020)
* Project:        basic_arith_ap_proj
* Solution:       solution1
* Product family: virtexuplus
* Target device:  xcvu9p-flga2104-1-i


================================================================
== Performance Estimates
================================================================
+ Timing:
    * Summary:
    +--------+---------+---------+------------+
    |  Clock |  Target | Estimated| Uncertainty|
    +--------+---------+---------+------------+
    |ap_clk  | 25.00 ns | 2.846 ns |   3.12 ns  |
    +--------+---------+---------+------------+
```

```
+ Latency:
    * Summary:
    +---------+---------+----------+----------+-----+-----+---------+
    |  Latency (cycles) |  Latency (absolute) |  Interval | Pipeline|
    |   min   |   max   |    min   |    max   | min | max |   Type  |
    +---------+---------+----------+----------+-----+-----+---------+
    |       36|       36| 0.900 us | 0.900 us |   36|   36|   none  |
    +---------+---------+----------+----------+-----+-----+---------+
```

```
================================================================
== Utilization Estimates
================================================================
* Summary:
+---------------------+---------+-------+---------+---------+-----+
|         Name        | BRAM_18K| DSP48E|    FF   |   LUT   | URAM|
+---------------------+---------+-------+---------+---------+-----+
|DSP                  |        -|      1|        -|        -|    -|
|Expression           |        -|      -|        0|       20|    -|
|FIFO                 |        -|      -|        -|        -|    -|
|Instance             |        -|      -|      680|      395|    -|
|Memory               |        -|      -|        -|        -|    -|
|Multiplexer          |        -|      -|        -|      169|    -|
|Register             |        -|      -|       37|        -|    -|
+---------------------+---------+-------+---------+---------+-----+
|Total                |        0|      1|      717|      584|    0|
+---------------------+---------+-------+---------+---------+-----+
|Available SLR        |     1440|   2280|   788160|   394080|  320|
+---------------------+---------+-------+---------+---------+-----+
|Utilization SLR (%)  |        0|   ~0  |    ~0   |    ~0   |    0|
+---------------------+---------+-------+---------+---------+-----+
|Available            |     4320|   6840|  2364480|  1182240|  960|
+---------------------+---------+-------+---------+---------+-----+
|Utilization (%)      |        0|   ~0  |    ~0   |    ~0   |    0|
+---------------------+---------+-------+---------+---------+-----+
```

Data-Type (w/ Arbitrary precision)

+ Detail:
  * Instance:

Instance	Module	BRAM_18K	DSP48E	FF	LUT	URAM
basic_arith_ap_sdcud_U2	basic_arith_ap_sdcud	0	0	274	148	0
basic_arith_ap_srbkb_U1	basic_arith_ap_srbkb	0	0	406	247	0
Total		0	0	680	395	0

  * DSP48E:

Instance	Module	Expression
basic_arith_ap_mudEe_U3	basic_arith_ap_mudEe	i0 * i1

  * Expression:

Variable Name	Operation	DSP48E	FF	LUT	Bitwidth P0	Bitwidth P1
out2_V	+	0	0	20	13	13
Total		0	0	20	13	13

  * Multiplexer:

Name	LUT	Input Size	Bits	Total Bits
ap_NS_fsm	169	38	1	38
Total	169	38	1	38

  * Register:

Name	FF	LUT	Bits	Const Bits
ap_CS_fsm	37	0	37	0
Total	37	0	37	0

== Interface

* Summary:

RTL Ports	Dir	Bits	Protocol	Source Object	C Type
ap_clk	in	1	ap_ctrl_hs	basic_arith_ap	return value
ap_rst	in	1	ap_ctrl_hs	basic_arith_ap	return value
ap_start	in	1	ap_ctrl_hs	basic_arith_ap	return value
ap_done	out	1	ap_ctrl_hs	basic_arith_ap	return value
ap_idle	out	1	ap_ctrl_hs	basic_arith_ap	return value
ap_ready	out	1	ap_ctrl_hs	basic_arith_ap	return value
inA_V	in	6	ap_none	inA_V	scalar
inB_V	in	12	ap_none	inB_V	scalar
inC_V	in	22	ap_none	inC_V	scalar
inD_V	in	33	ap_none	inD_V	scalar
out1_V	out	18	ap_vld	out1_V	pointer
out1_V_ap_vld	out	1	ap_vld	out1_V	pointer
out2_V	out	13	ap_vld	out2_V	pointer
out2_V_ap_vld	out	1	ap_vld	out2_V	pointer
out3_V	out	22	ap_vld	out3_V	pointer
out3_V_ap_vld	out	1	ap_vld	out3_V	pointer
out4_V	out	6	ap_vld	out4_V	pointer
out4_V_ap_vld	out	1	ap_vld	out4_V	pointer

TAC-HEP 2025

HLS Pragmas

HLS Pragmas

HLS pragmas are compiler directives used in HLS tools (like Xilinx Vitis/Vivado HLS or Intel HLS compiler) to optimize hardware implementation while writing high-level C, C++ or SystemC code

HLS tool provides pragmas that can be used to

- Optimize the design
- Reduce latency
- Improve throughput performance
- Reduce area and device resource usage

List of Available Pragmas

Type	Attributes
Kernel Optimization	• pragma HLS aggregate • pragma HLS alias • pragma HLS disaggregate • pragma HLS expression_balance • pragma HLS latency • pragma HLS performance • pragma HLS protocol • pragma HLS reset • pragma HLS top • pragma HLS stable
Function Inlining	• pragma HLS inline
Interface Synthesis	• pragma HLS interface • pragma HLS stream
Task-level Pipeline	• pragma HLS dataflow • pragma HLS stream
Pipeline	• pragma HLS pipeline • pragma HLS occurrence
Loop Unrolling	• pragma HLS unroll • pragma HLS dependence
Loop Optimization	• pragma HLS loop_flatten • pragma HLS loop_merge • pragma HLS loop_tripcount
Array Optimization	• pragma HLS array_partition • pragma HLS array_reshape
Structure Packing	• pragma HLS aggregate • pragma HLS dataflow
Resource Utilization	• pragma HLS allocation • pragma HLS bind_op • pragma HLS bind_storage • pragma HLS function_instantiate

Pragma HLS interface

- **C/C++ based design:** Input & outputs are performed in zero time through function arguments
- **RTL design:** same I/O operations must be performed through a port in the design interface & typically operates using a specific I/O protocol
- INTERFACE pragma specifies how RTL ports are created from the function definitions during interface synthesis

https://docs.amd.com/r/en-US/ug1399-vitis-hls/pragma-HLS-interface

Pragma HLS interface

- The **INTERFACE pragma** or directive is only supported for use on the top-level function, and cannot be used for sub-functions of the HLS component
  - HLS tool automatically determines the I/O protocol used by any sub-functions

- The arguments of the top-level function in an HLS component are synthesized into interfaces and ports that group multiple signals to define the communication protocol between the HLS component and elements external to the design

- The type of interfaces that the tool chooses depends on the data type and direction of the parameters of the top-level function, the target flow for the HLS component

Interface

The interface defines three elements of the kernel:

- The interface defines channels for data to flow into or out of the HLS design. Data can flow from a variety of sources external to the kernel or IP, such as a host application, an external camera or sensor, or from another kernel or IP implemented on the AMD device

- The interface defines the port protocol that is used to control the flow of data through the data channel, defining when the data is valid and can be read or can be written

- The interface also defines the execution control scheme for the HLS design, specifying the operation of the kernel or IP as pipelined or sequential

Control signals: ap_start

- This signal controls the block execution and must be asserted to ***logic 1*** for the design to begin operation.
- It should be held at ***logic 1*** until the associated output handshake *ap_ready* is asserted.
- Keep *ap_start* = 1 until ap_ready becomes **1** (meaning the task is done, and new data can be processed)
- If *ap_start* is asserted low before *ap_ready* is high, the design might not have read all input ports and might stall operation on the next input read

Control Signal: ap_ready

- This output signal indicates when the design is ready for new inputs.
- The *ap_ready* signal is set to **logic 1** when the design is ready to accept new inputs, indicating that all input reads for this transaction have been completed.
- If the design has no pipelined operations, new reads are not performed until the next transaction starts.
- If *ap_start = 0*, the design will **stop** after finishing its current task.

Control Signal: ap_done

- This signal indicates when the design has completed all operations in the current transaction.
- *ap_done* = 1 means the design has **finished processing** all operations for the current task.
- If there is an *ap_return* output, the value is now **valid** and ready to be read.
- Not all functions have a function return argument and hence not all RTL designs have an ap_return port

Control Signal: ap_idle

- This signal indicates if the design is operating or idle (no operation).
- What *ap_idle* = 1 means the design is not doing anything (idle), It is waiting for *ap_start* = 1 to begin working
- This signal is asserted high when the design completes operation and no further operations are performed.

TAC-HEP 2025

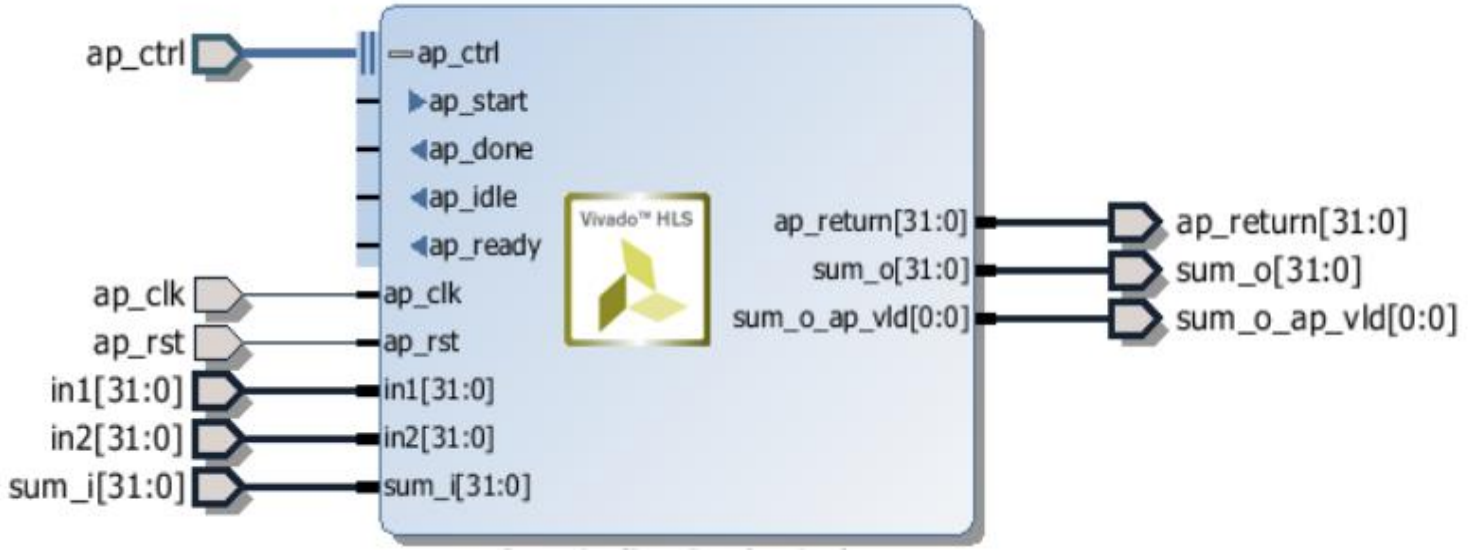
HLS Interface Example

Interface Synthesis overview

```
#include "sum_io.h"
dout_t sum_io(din_t in1, din_t in2, dio_t *sum) {
    dout_t temp;
    *sum = in1 + in2 + *sum;
    temp = in1 + in2;
    return  temp;
}
```

- Two inputs *n1* & *n2*
- A pointer *sum* that is read from and written to
- A function *return*, the value of *temp*

Default interface settings will synthesize the design into a RTL block with ports as shown:



Interface Synthesis overview

Three types of ports in the design:

- **Clock & reset ports**: *ap_clk* and *ap_rst*
  - If the design takes more than 1 clock cycle to complete

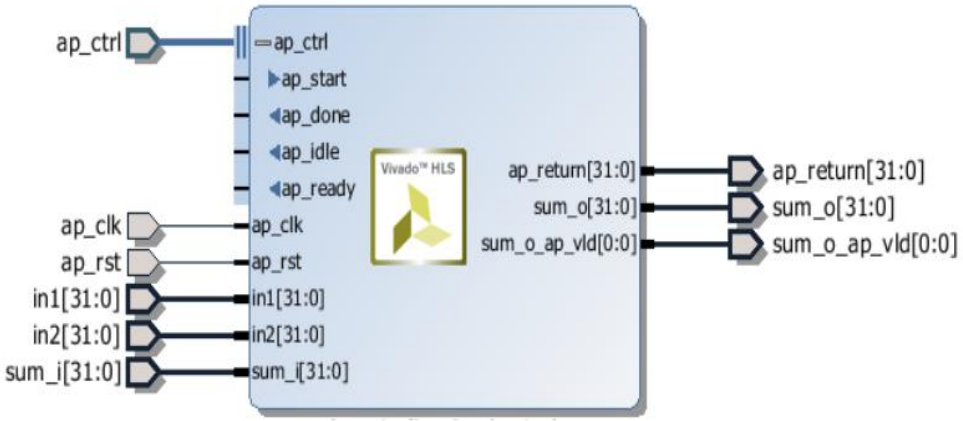
- **Block-level interface protocol:**
  - Added by default & control the block
  - Independent to anyport-level protocol
  - *ap_start*: Control when block can start processing data
  - *ap_ready*: when ready to accept new input
  - *ap_idle*: if the design is idle
  - *ap_done*: completed operation

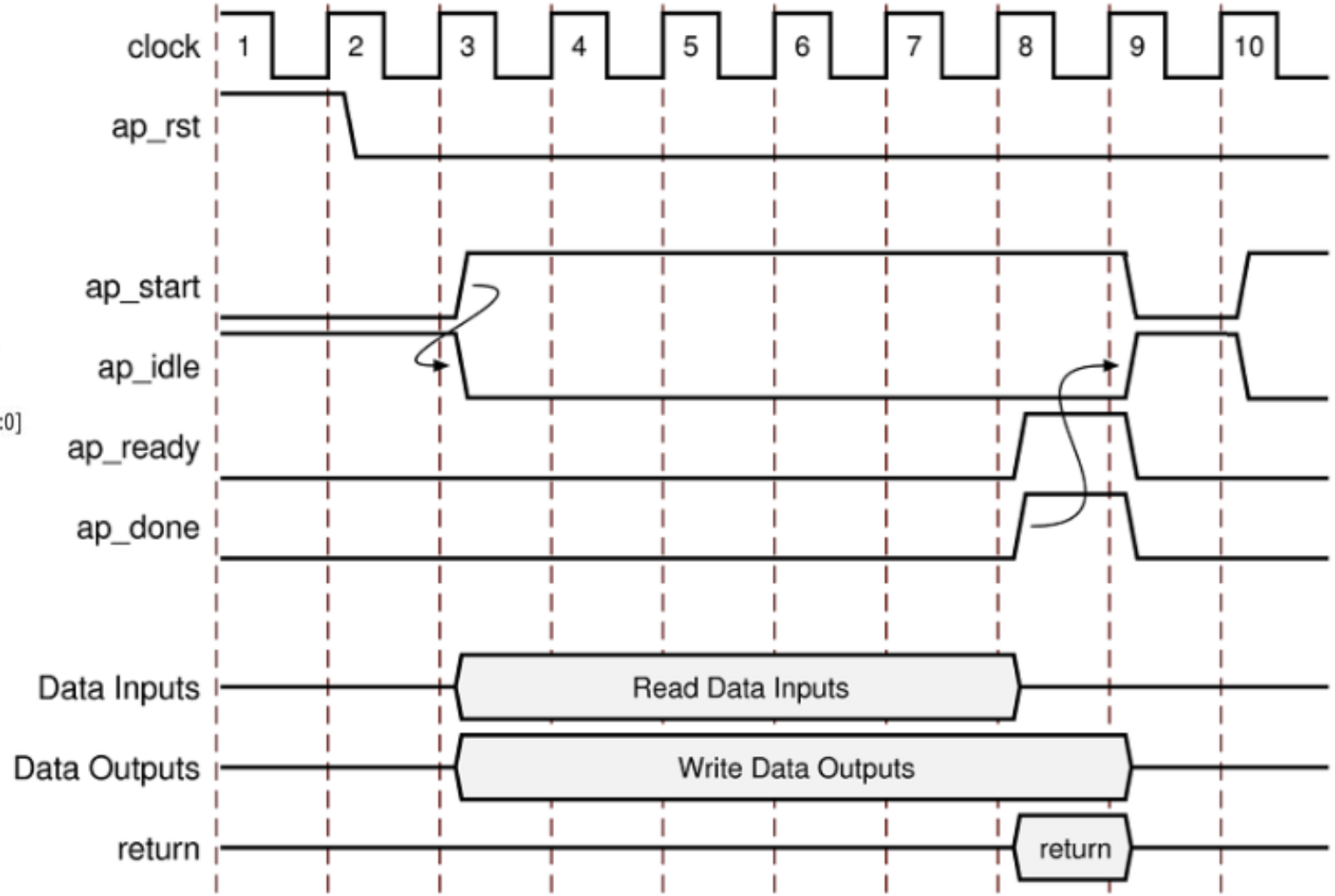
- **Port level interface portocols:** *in1*, *in2*, *sum_i*, *sum_o*, *sum_o_ap_vld*, and *ap_return*
  - Final group of signals
  - Created for each argument in the top-level function & the function return
  - After block-level protocol has been used to start the operation of block, port level I/O protocols are used to sequence data in and out of the block

Port-Level Interface Protocol

- By default, input pass-by-value arguments and pointers are implemented as simple wire ports with no associated handshaking signal
  - Ex: Input ports are implemented without an I/O protocol, only a data port (data is held stable until it is read)
- By default, output pointers are implemented with an associated output valid signal *(sum_o_ap_vld)* to indicate when the output data is valid
  - No I/O protocol associated with the output port, it is difficult to know when to read the data
  - It is always a good idea to use an I/O protocol on an output
- Function arguments that are both read from & writes to are split into separate input & output ports
  - Ex: sum is implemented as input port sum_i and output port sum_o with associated I/O protocol port sum_o_ap_vld
- Function with a return value, an output port *ap_return* is implemented to provide the return value
- Completion of one transaction: the block-level protocols indicate the function is complete with the *ap_done* signal.
  - Also indicates the data on port *ap_return* is valid and can be read

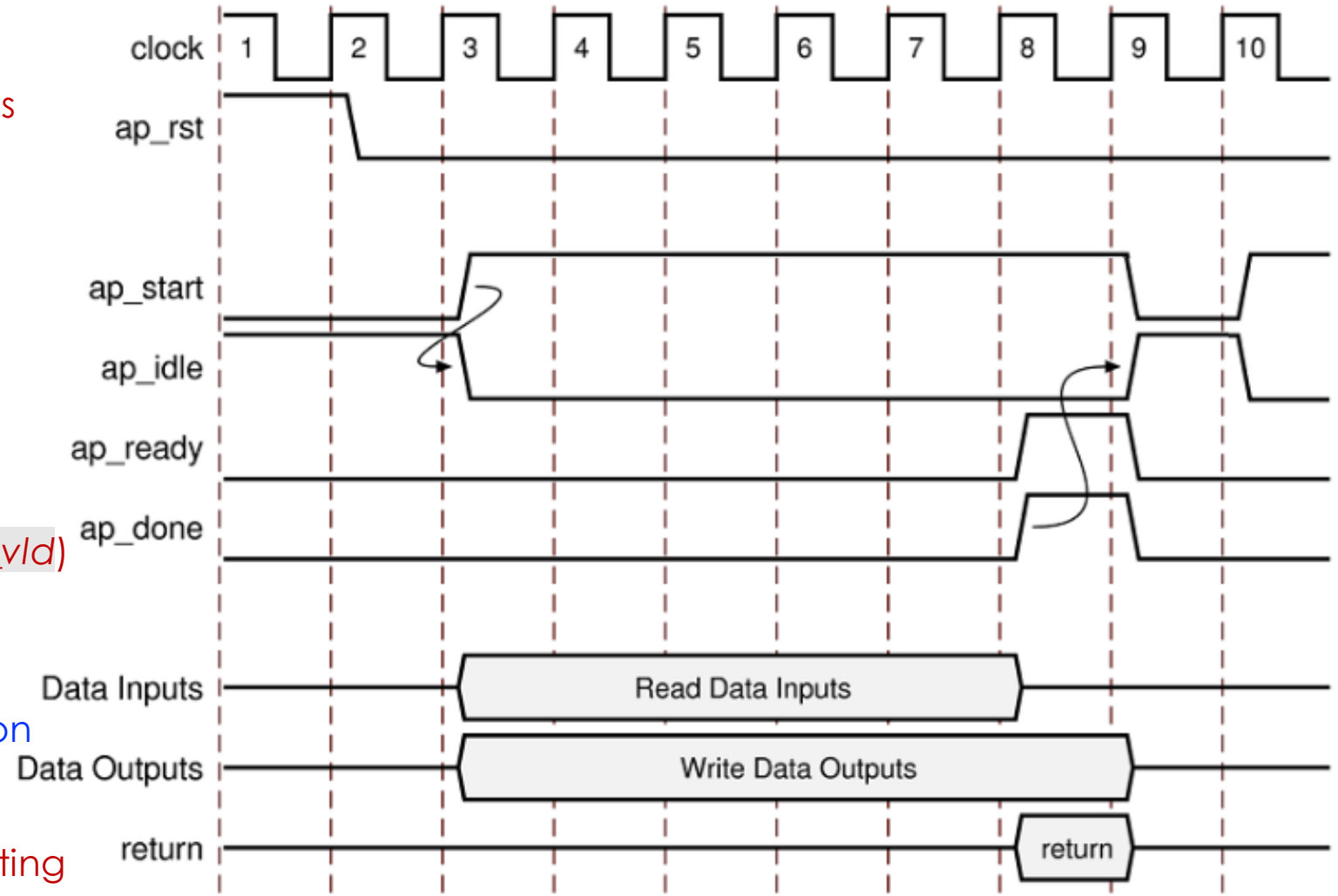
RTL Port timing





RTL Port timing

- Design starts: *ap_start* is High
- *ap_idle* signal goes Low indicating design is operating
- Input data is read at any CLK after the first cycl.
- HLS schedules when the reads occur
- *ap_ready* signal is asserted high when all inputs have been read
- When output sum is calculated, the associated output handshake (*sum_o_ap_vld*) indicates that the data is valid
- When the function completes, *ap_done* is asserted. This also indicates that the data on *ap_return* is valid
- Port *ap_idle* is High indicating design is waiting start again



Interface Synthesis I/O Protocols

The type of interfaces that are created by interface synthesis depends on the type of C/C++ argument

D: Default interface mode for each type

I: Input arguments, which are only read

O: Output arguments, which are only written to

I/O: Input/Output arguments, which are both read and written

Argument Type	Scalar		Array			Pointer or Reference			HLS:: Stream
Interface Mode	Input	Return	I	I/O	O	I	I/O	O	I and O
ap_ctrl_none									
ap_ctrl_hs		D							
ap_ctrl_chain									
axis									
s_axilite									
m_axi									
ap_none	D					D			
ap_stable									
ap_ack									
ap_vld								D	
ap_ovld							D		
ap_hs									
ap_memory			D	D	D				
bram									
ap_fifo									D
ap_bus									

Supported D = Default Interface Not Supported

X14293

Assignment #4

1.) Write a simple program doing arithmetic operations (+, -, *, /, %) between elements of two arrays (N > 10) using arbitrary precision and compare results with standard c/c++ data types and using ap_(u)int<N>

Share the detail comparison report of the two and your conclusion

Reminder: Assignments

- Assignment-1 (13-02-2025)
- Assignment-2 (18-02-2025)
- Assignment-3 (27-02-2025)
- Assignment-4 (06-03-2025)

Uploaded to cernbox: https://cernbox.cern.ch/s/gmUqRDHTxDLqx4M

Send via email: **varun.sharma@cern.ch**

Submit in 2 weeks from date of assignment

TAC-HEP 2025

Questions?

Acknowledgements:
- https://docs.amd.com/r/en-US/ug1399-vitis-hls/HLS-Pragmas
- ug871-vivado-high-level-synthesis-tutorial.pdf

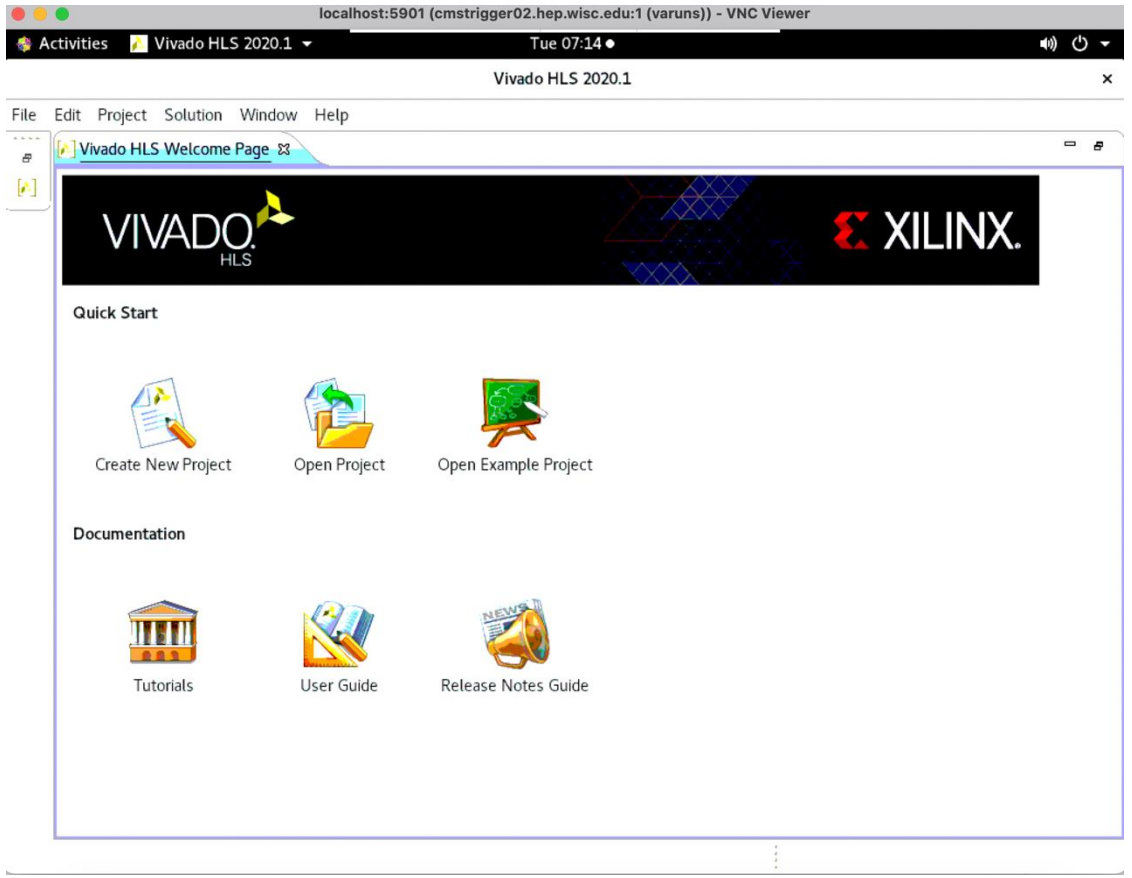
Reminder: HLS Setup

- ssh <username>@cmstrigger02-via-login -L5901:localhost:5901
  - Or whatever **:1** display number
  - Sometimes you may need to run vncserver -localhost -geometry 1024x768 again to start new vnc server

- **Connect to VNC server (remote desktop) client**
- **Open terminal**
  - source /opt/Xilinx/Vivado/2020.1/settings64.sh
  - cd /scratch/`whoami`
  - vivado_hls

OR

  - Source /opt/Xilinx/Vitis/2020.1/settings64.sh
  - Cd /scratch/`whoami`
  - vitis_hls



Jargons

- **ICs - Integrated chip:** assembly of hundreds of millions of transistors on a minor chip
- **PCB:** Printed Circuit Board
- **LUT - Look Up Table aka 'logic'** - generic functions on small bitwidth inputs. Combine many to build the algorithm
- **FF - Flip Flops** - control the flow of data with the clock pulse. Used to build the pipeline and achieve high throughput
- **DSP - Digital Signal Processor** - performs multiplication and other arithmetic in the FPGA
- **BRAM - Block RAM** - hardened RAM resource. More efficient memories than using LUTs for more than a few elements
- **PCIe or PCI-E - Peripheral Component Interconnect Express**: is a serial expansion bus standard for connecting a computer to one or more peripheral devices
- **InfiniBand** is a computer networking communications standard used in high-performance computing that features very high throughput and very low latency
- **HLS** - High Level Synthesis - compiler for C, C++, SystemC into FPGA IP cores
- **HDL** - Hardware Description Language - low level language for describing circuits
- **RTL** - Register Transfer Level - the very low level description of the function and connection of logic gates
- **FIFO** – First In First Out memory
- **Latency** - time between starting processing and receiving the result
  - Measured in clock cycles or seconds
- **II - Initiation Interval** - time from accepting first input to accepting next input