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

FPGA module training

Lecture-7: March 17th 2026

Varun Sharma

University of Wisconsin – Madison, USA

Content

So far

- Motivation & Introduction
- Comparison: FPGAs/ASICs/GPU/CPU
- Domain specific Accelerators
- HLS setup and first example project
- Unsupported C/C++ constructs
- Data types

Today

- HLS Pragmas
  - Interface
  - Array partition
  - Array Reshape

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

HLS Pragmas

"Pragmas": Instructions to tell your compiler how to build the hardware

- HLS tool provides different set of pragmas that can be used to optimize the design, reduce latency, improve performance etc. These pragmas can be directly added to the source code for the kernel.

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

https://docs.xilinx.com/r/en-US/ug1399-vitis-hls/HLS-Pragmas

TAC-HEP 2026

Pragma HLS Interface

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
  - Bridge between your C/C++ hardware functions and the outside world (like DDR memory, CPUs, or other IP cores)
  - In HLS, your function arguments don't automatically know how they should physically connect to the FPGA pins
  - this pragma tells the compiler exactly which hardware protocol to use

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

Pragma HLS interface: Syntax

#pragma HLS interface mode=<mode> port=<name> direct_io=<value>**[OPTIONS]**

mode = <mode>

- Can be broken into three categories
  1. Port-level Protocols
  2. AXI Interface Protocols
  3. Block-Level Control Ports

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

Key Interfaces

HLS supports memory, stream, and register interface paradigms where each paradigm follows a certain interface protocol and uses the adapter to communicate with the external world

- **Register Paradigm (s_axilite):**
  - Small, register-based interface used for control signals (start, stop, interrupt) and simple scalar parameters.
- **Memory Paradigm (m_axi):**
  - Memory-mapped interface used for high-bandwidth data transfers, reading/writing to DDR, HBM, PLRAM/BRAM/URAM
- **Stream Paradigm (axis):**
  - A streaming interface (AXI4-Stream) for continuous data flow without addresses such as video processors
- **ap_none/ap_vld:**
  - Simple “wire” interfaces with or without a valid signal

Default Interfaces

C-Argument Type	Supported Paradigms	Default Paradigm	Default Interface Protocol		
			Input	Output	Inout
Scalar variable (pass by value)	Register	Register	ap_none	N/A	N/A
Array	Memory, Stream	Memory	ap_memory	ap_memory	ap_memory
Pointer	Memory, Stream, Register	Register	ap_none	ap_vld	ap_ovld
Reference	Register	Register	ap_none	ap_vld	ap_vld
`hls::stream`	Stream	Stream	ap_fifo	ap_fifo	N/A

Interface overview

Function arguments in HLS are synthesized into **interfaces and ports**

- These group multiple signals and define **communication protocols**.
- **The goal:** connect HLS components to external elements
  - Example: memory, host, sensors

Interfaces define 3 key aspects of an HLS kernel:

1. **Data Channels:**
   - Enable data transfer into/out of the HLS design
   - **Sources:** host application, sensors, external IPs, or other kernels.
   - Default: AXI adapters (e.g., AXI4-Stream, AXI4-Memory).
2. **Port-Level Protocols:**
   - Control **when** data is valid for read/write.
   - Protocols manage flow using signals like *TVALID*, *TREADY*, etc.
   - Can be customized in **Vivado IP flow**, fixed in **Vitis kernel flow**.
3. **Execution Control Scheme:**
   - Defines kernel/IP operation as **pipelined** or **sequential**.
   - Controlled by block-level protocols.

Pragma HLS interface: Syntax

#pragma HLS interface mode=<mode> port=<name> direct_io=<value>**[OPTIONS]**

<mode>: Port-Level Protocols

- ***ap_none:*** No protocol. The interface is a data port
- ***ap_vld:*** Implements the data port with an associated valid port to indicate when the data is valid for reading or writing

- ***ap_ack:*** Implements the data port with an associated acknowledge port to acknowledge that the data was read or written
- ***ap_hs:*** Implements the data port with associated valid and acknowledge ports to provide a two-way handshake to indicate when the data is valid for reading and writing and to acknowledge that the data was read or written

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

Pragma HLS interface: Syntax

#pragma HLS interface mode=<mode> port=<name> direct_io=<value>**[OPTIONS]**

<mode>: Port-Level Protocols

- ***ap_stable*:** No protocol. The interface is a data port. The HLS tool assumes the data port is always stable after reset, which allows internal optimizations to remove unnecessary registers
- ***ap_fifo*:** Implements the port with a standard FIFO interface using data I/O ports with associated active-Low FIFO empty and full ports

- *ap_bus*: Implements pointer and pass-by-reference ports as a bus interface.
- *ap_memory*: Implements array arguments as a standard RAM interface
- ***ap_ovld*:** Implements the output data port with an associated valid port to indicate when the data is valid for reading or writing

Pragma HLS interface: Syntax

#pragma HLS interface mode=<mode> port=<name> direct_io=<value>**[OPTIONS]**

<mode>: AXI-Interface Protocols

- ***s_axilite***: Implements all ports as an AXI4-Lite interface. The tool produces an associated set of C driver files when exporting the generated RT for the HLS component
- ***m_axi*:** Implements all ports as an AXI4 interface
- ***m_axi_addr64*** command to specify either 32-bit (default) or 64-bit address ports and to control any address offset.
- ***axis***: Implements all ports as an AXI4-Stream interface

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

Pragma HLS interface: Syntax

#pragma HLS interface mode=<mode> port=<name> direct_io=<value>**[OPTIONS]**

<mode>: Block-level Control Protocols

- ***ap_ctrl_chain*:** Implements a set of block-level control ports to **start** the design operation, **continue** operation & indicate when the design is **idle**, **done**, & **ready** for new input data
- ***ap_ctrl_none***: No block-level I/O protocol
- ***ap_ctrl_hs***: Implements a set of block-level control ports to start the design operation and to indicate when the design is idle, done, and ready for new input data

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

Pragma HLS interface: Syntax

#pragma HLS interface mode=<mode> port=<name> direct_io=<value>**[OPTIONS]**

port=<name>: Specifies the name of the function argument, function return, or global variable which the INTERFACE pragma applies to

[OPTIONS]
bundle=<string>: Groups function arguments into AXI interface ports

register: An optional keyword to register the signal and any relevant protocol signals, and causes the signals to persist until at least the last cycle of the function execution.

- Ap_none, ap_ack, ap_vld, ap_ovld, ap_hs, ap_stable, axis, s_axilite

TAC-HEP 2026

Some examples

Example – 1

```
void example(int a, int b, int *c){

    #pragma HLS INTERFACE s_axilite port=a
    #pragma HLS INTERFACE s_axilite port=b
    #pragma HLS INTERFACE s_axilite port=c
    #pragma HLS INTERFACE s_axilite port=return

    *c = a + b;
}
```

```
================================================================
== Utilization Estimates
================================================================
* Summary:
+---------------------+---------+-------+---------+---------+-----+
|         Name        | BRAM_18K| DSP48E|    FF   |   LUT   | URAM|
+---------------------+---------+-------+---------+---------+-----+
|DSP                  |        -|      -|        -|        -|    -|
|Expression           |        -|      -|        0|       39|    -|
|FIFO                 |        -|      -|        -|        -|    -|
|Instance             |        0|      -|      150|      232|    -|
|Memory               |        -|      -|        -|        -|    -|
|Multiplexer          |        -|      -|        -|        -|    -|
|Register             |        -|      -|        -|        -|    -|
+---------------------+---------+-------+---------+---------+-----+
|Total                |        0|      0|      150|      271|    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|
+---------------------+---------+-------+---------+---------+-----+
```

Example – 2

```
void example(int *input, int *output, int size) {
   #pragma HLS INTERFACE m_axi port=input depth=1024
   #pragma HLS INTERFACE m_axi port=output depth=1024
   #pragma HLS INTERFACE s_axilite port=size
   #pragma HLS INTERFACE s_axilite port=return


   for (int i = 0; i < size; i++) {
      output[i] = input[i] * 2;
   }
}
```

```
================================================================
== Utilization Estimates
================================================================
* Summary:
+---------------------+---------+------+---------+---------+-----+
|         Name        | BRAM_18K| DSP48E|    FF   |   LUT   | URAM|
+---------------------+---------+------+---------+---------+-----+
|DSP                  |        -|     -|        -|        -|    -|
|Expression           |        -|     -|        0|       60|    -|
|FIFO                 |        -|     -|        -|        -|    -|
|Instance             |        4|     -|     1098|     1264|    -|
|Memory               |        -|     -|        -|        -|    -|
|Multiplexer          |        -|     -|        -|      116|    -|
|Register             |        -|     -|      140|        -|    -|
+---------------------+---------+------+---------+---------+-----+
|Total                |        4|     0|     1238|     1440|    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|
+---------------------+---------+------+---------+---------+-----+
```

Example – 3

```
#include <hls_stream.h>

void example(hls::stream<int> &input, hls::stream<int> &output)
{
  #pragma HLS INTERFACE axis port=input
  #pragma HLS INTERFACE axis port=output
  #pragma HLS INTERFACE ap_ctrl_none port=return

  int data;
  if (input.read_nb(data)) { // Non-blocking read
     output.write(data * 2);
  }
}
```

```
================================================================
== Utilization Estimates
================================================================
* Summary:
+---------------------+---------+-------+---------+---------+-----+
|         Name        | BRAM_18K| DSP48E|    FF   |   LUT   | URAM|
+---------------------+---------+-------+---------+---------+-----+
|DSP                  |        -|      -|        -|        -|    -|
|Expression           |        -|      -|        0|        6|    -|
|FIFO                 |        -|      -|        -|        -|    -|
|Instance             |        -|      -|        -|        -|    -|
|Memory               |        -|      -|        -|        -|    -|
|Multiplexer          |        -|      -|        -|       24|    -|
|Register             |        -|      -|        3|        -|    -|
+---------------------+---------+-------+---------+---------+-----+
|Total                |        0|      0|        3|       30|    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|
+---------------------+---------+-------+---------+---------+-----+
```

Example – 4

```
void example(int input[256], int output[256]) {

   #pragma HLS INTERFACE bram port=input
   #pragma HLS INTERFACE bram port=output
   #pragma HLS INTERFACE s_axilite port=return

   for (int i = 0; i < 256; i++) {
      output[i] = input[i] * 2;
   }
}
```

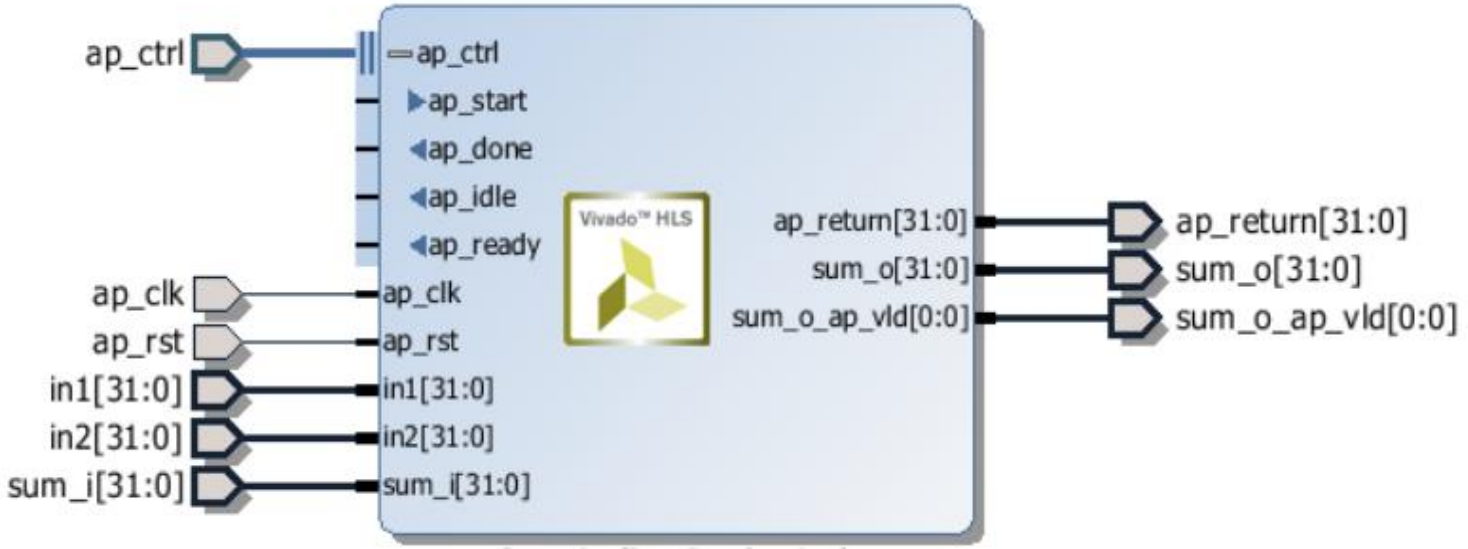
```
================================================================
== Utilization Estimates
================================================================
* Summary:
+---------------------+---------+------+---------+---------+-----+
|         Name        | BRAM_18K| DSP48E|    FF   |   LUT   | URAM|
+---------------------+---------+------+---------+---------+-----+
|DSP                  |        -|     -|        -|        -|    -|
|Expression           |        -|     -|        0|       29|    -|
|FIFO                 |        -|     -|        -|        -|    -|
|Instance             |        0|     -|       36|       40|    -|
|Memory               |        -|     -|        -|        -|    -|
|Multiplexer          |        -|     -|        -|       39|    -|
|Register             |        -|     -|       30|        -|    -|
+---------------------+---------+------+---------+---------+-----+
|Total                |        0|     0|       66|      108|    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|
+---------------------+---------+------+---------+---------+-----+
```

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:



Defining Interfaces: examples

Function arguments in HLS are synthesized into **interfaces and ports**

- These group multiple signals and define **communication protocols**.
- **The goal:** connect HLS components to external elements
  - Example: memory, host, sensors

Image processing from a camera

- Apply a filter to live video
- Camera sends pixel data via **AXI-4 stream** to an HLS block
- Processes image in real time
- Output stream sent to display or written to memory

HLS block connected to **external camera (sensor)** via streaming interface

Accelerating Matrix Multiplication with external DDR

- Large matrix multiplication that can't fit on-chip
- Host application sends matrices to **external DDR memory**
- Reads matrices using **AXI-4 memory** mapped interface
- Performs multiplication
- Writes result back to memory

HLS block connected to **DDR memory** through an AXI interface

Defining Interfaces: examples

Function arguments in HLS are synthesized into **interfaces and ports**

- These group multiple signals and define **communication protocols**.
- **The goal:** connect HLS components to external elements
  - Example: memory, host, sensors

Data Transfer from Host CPU

- Offload compute-heavy operation (e.g. encryption) to FPGA
- Sends data via PCIe to FPGA
- HLS kernel receives data through **AXI-4 stream**
- Processes encription
- Sends results back to host

HLS block interfaces using **AXI4-stream** and **AXI-Lite** (for control)

TAC-HEP 2026

Arrays

Arrays

- Fundamental data structure in any C++ software program
  - Simple containers
  - Often dynamically allocated/deallocated on demand

- Hardware:
  - Dynamic memory allocation is NOT supported
  - Exact amount of memory required
  - Typically implemented as memory (RAM, ROM, or shifters) after synthesis
  - Arrays can be partitioned into blocks or into their individual elements

Array Accesses & Performance

- Array access patterns can affect the performance when arrays are mapped to memories instead of registers
- **Registers:** directly accessible & provide fast access, minimal latency
- **Memories:** like RAM or external memory, introduce variable access time, higher latency
- Arrays in **memory** may have data dependencies, where one access depends on the result of another.
- **Registers** are typically managed to minimize such dependencies

Example-1 (ex-arr-mem)

```
#include "example.h"

dout_t example(din_t mem[N]) {

    dout_t total = 0;

    for (int i = 2; i < N; ++i)
        total += mem[i] + mem[i - 1] + mem[i - 2];

    return total;

}
```

+ Timing:
  * Summary:

Clock	Target	Estimated	Uncertainty
ap_clk	25.00 ns	3.390 ns	3.12 ns

+ Latency:
  * Summary:

Latency (cycles)		Latency (absolute)		Interval		Pipeline
min	max	min	max	min	max	Type
187	187	4.675 us	4.675 us	187	187	none

  * Loop:

Loop Name	Latency (cycles)		Iteration	Initiation Interval		Trip	
	min	max	Latency	achieved	target	Count	Pipelined
- Loop 1	186	186	3	-	-	62	no

Example-1 (ex-arr-mem)

```
#include "example.h"

dout_t example(din_t mem[N]) {

    dout_t total = 0;

    for (int i = 2; i < N; ++i)
        total += mem[i] + mem[i - 1] + mem[i - 2];

    return total;

}
```

```
================================================================
== Utilization Estimates
================================================================
* Summary:
+---------------------+---------+-------+---------+---------+-----+
|         Name        | BRAM_18K| DSP48E|    FF   |   LUT   | URAM|
+---------------------+---------+-------+---------+---------+-----+
|DSP                  |        -|      -|        -|        -|    -|
|Expression           |        -|      -|        0|      104|    -|
|FIFO                 |        -|      -|        -|        -|    -|
|Instance             |        -|      -|        -|        -|    -|
|Memory               |        -|      -|        -|        -|    -|
|Multiplexer          |        -|      -|        -|       60|    -|
|Register             |        -|      -|       35|        -|    -|
+---------------------+---------+-------+---------+---------+-----+
|Total                |        0|      0|       35|      164|    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|
+---------------------+---------+-------+---------+---------+-----+
```

Example-2 (ex-arr-mem-perf)

Let's restructure the code to pipeline a bit manually

```
#include "example.h"

dout_t example(din_t mem[N]) {

    din_t temp0, temp1, temp2;
    dout_t total = 0;

    temp0 = mem[0];
    temp1 = mem[1];
    for (int i = 2; i < N; ++i){
        temp2 = mem[i];
        total += temp0 + temp1 +
temp2;
        temp0 = temp1;
        temp1 = temp2;

    }

    return total;
}
```

Example-2 (ex-arr-mem-perf)

Results

```
#include "example.h"

dout_t example(din_t mem[N]) {

   din_t temp0, temp1, temp2;
   dout_t total = 0;

   temp0 = mem[0];
   temp1 = mem[1];
   for (int i = 2; i < N; ++i){
      temp2 = mem[i];
      total += temp0 + temp1 +
temp2;
      temp0 = temp1;
      temp1 = temp2;

   }

   return total;
}
```

```
+ Timing:
    * Summary:
    +--------+----------+----------+------------+
    |  Clock |  Target  | Estimated| Uncertainty|
    +--------+----------+----------+------------+
    |ap_clk  | 25.00 ns | 2.534 ns |   3.12 ns  |
    +--------+----------+----------+------------+

+ Latency:
    * Summary:
    +---------+---------+----------+----------+-----+-----+---------+
    |  Latency (cycles) |  Latency (absolute) |  Interval | Pipeline|
    |   min   |   max   |    min   |    max   | min | max |   Type  |
    +---------+---------+----------+----------+-----+-----+---------+
    |      126|      126| 3.150 us | 3.150 us |  126|  126|   none  |
    +---------+---------+----------+----------+-----+-----+---------+

* Loop:
+----------+---------+---------+----------+-----------+-----------+------+----------+
|          |  Latency (cycles) | Iteration|  Initiation Interval  | Trip |          |
| Loop Name|   min   |   max   |  Latency |  achieved |   target  | Count| Pipelined|
+----------+---------+---------+----------+-----------+-----------+------+----------+
|- Loop 1  |      124|      124|         2|          -|          -|    62|    no    |
+----------+---------+---------+----------+-----------+-----------+------+----------+
```

Example-2 (ex-arr-mem-perf)

Results

```
#include "example.h"

dout_t example(din_t mem[N]) {

   din_t temp0, temp1, temp2;
   dout_t total = 0;

   temp0 = mem[0];
   temp1 = mem[1];
   for (int i = 2; i < N; ++i){
      temp2 = mem[i];
      total += temp0 + temp1 +
temp2;
      temp0 = temp1;
      temp1 = temp2;

   }

   return total;
}
```

```
================================================================
== Utilization Estimates
================================================================
* Summary:
+---------------------+---------+-------+---------+---------+-----+
|         Name        | BRAM_18K| DSP48E|    FF   |   LUT   | URAM|
+---------------------+---------+-------+---------+---------+-----+
|DSP                  |        -|      -|        -|        -|    -|
|Expression           |        -|      -|        0|       74|    -|
|FIFO                 |        -|      -|        -|        -|    -|
|Instance             |        -|      -|        -|        -|    -|
|Memory               |        -|      -|        -|        -|    -|
|Multiplexer          |        -|      -|        -|       78|    -|
|Register             |        -|      -|       50|        -|    -|
+---------------------+---------+-------+---------+---------+-----+
|Total                |        0|      0|       50|      152|    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|
+---------------------+---------+-------+---------+---------+-----+
```

Array Optimization Directives

Changes to the source code as shown above are not always required

The more typical case is to use optimization directives/pragmas to achieve the same result

Two main classes for optimization:

- **Array Partition:** splits apart original array into smaller arrays or individual registers
- **Array Reshape:** reorganizes the array into a different memory arrangement to increase parallelism but without splitting apart the original array

TAC-HEP 2026

#pragma HLS array_partition

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

Pragma HLS array_partition

- Partitions an array into smaller arrays or individual elements and provides the following:
  - Results in RTL with multiple small memories or multiple registers instead of one large memory
  - Effectively increases the amount of read and write ports for the storage
  - Potentially improves the throughput of the design
  - Requires more memory instances or registers

Syntax:

Place the pragma in the C source within the boundaries of the function where the array variable is defined

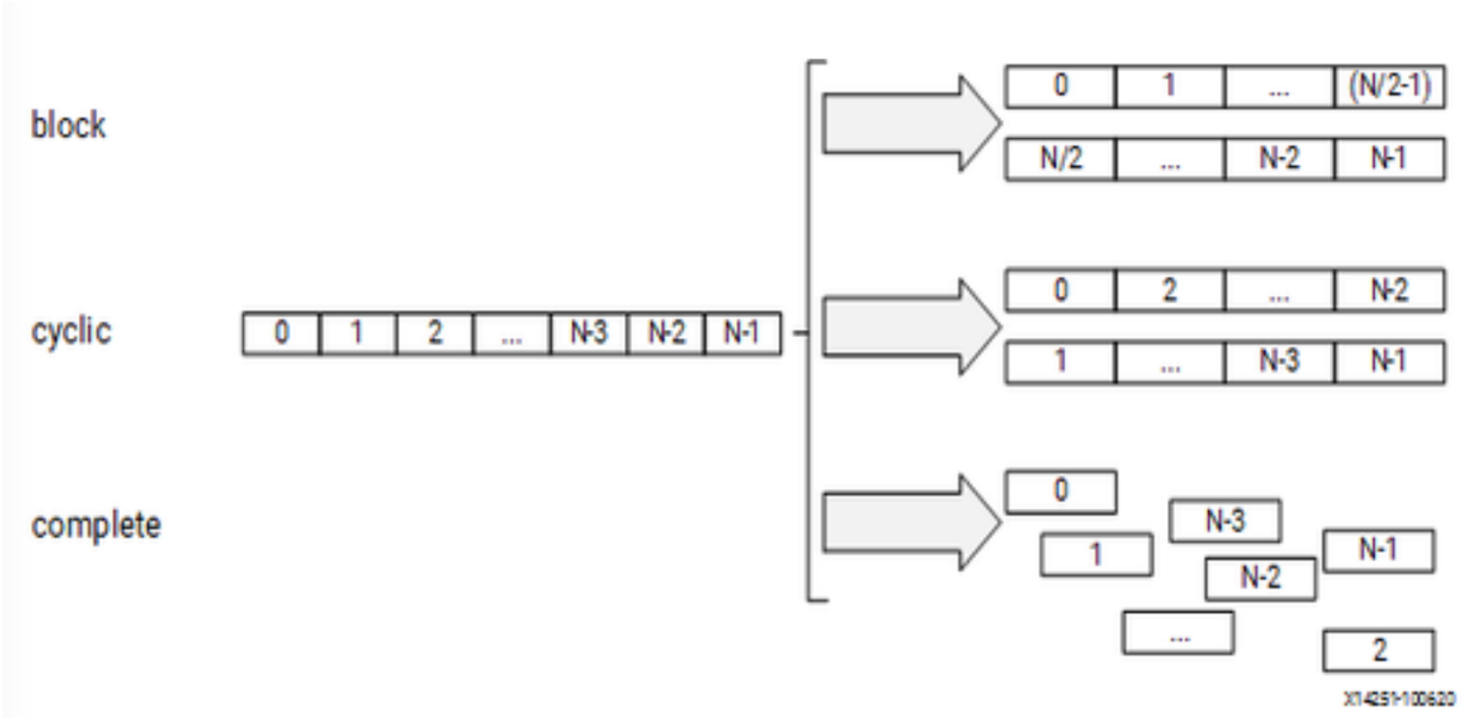
```
#pragma HLS array_partition variable=<name>  <type> factor=<int> dim=<int>
```

variable=<name>: A required argument that specifies the array variable to be partitioned

Pragma HLS array_partition

#pragma HLS array_partition variable=<name> <type> factor=<int> dim=<int>

Three ***types*** of array partitioning available:



Pragma HLS array_partition

#pragma HLS array_partition variable=<name> <type> factor=<int> dim=<int>

- **block:** Block partitioning creates smaller arrays from consecutive blocks of the original array
  - Effectively splits the array into N equal blocks, where N is the integer defined by the factor= argument

- **cyclic:** Cyclic partitioning creates smaller arrays by interleaving elements from the original array
  - Partitioned cyclically by putting one element into each new array before coming back to the first array to repeat the cycle until the array is fully partitioned.
  - For example, if factor=3 is used:
    - Element 0 is assigned to the first new array
    - Element 1 is assigned to the second new array.
    - Element 2 is assigned to the third new array.
    - Element 3 is assigned to the first new array again.

- **complete:** Complete partitioning decomposes the array into individual elements
  - For a 1-D array, this corresponds to resolving a memory into individual registers (default <type>)

Pragma HLS array_partition

#pragma HLS array_partition variable=<name> <type> factor=<int> dim=<int>

factor=<*int*>: Specifies the number of smaller arrays that are to be created

NOTE: For complete type partitioning, the factor is not specified. Must for *block* and *cyclic* partitioning

dim=<*int*>: Specifies which dimension of a multi-dimensional array to partition.
Specified as an integer from 0 to *N*, for an array with *N* dimensions:

- If a value of 0 is used, all dimensions of a multi-dimensional array are partitioned with the specified type and factor options.
- Any non-zero value partitions only the specified dimension
- **For example**, if a value 1 is used, only the first dimension is partitioned.

Partitioning Array Dimensions

```
#pragma HLS array_partition variable=Array complete dim=3
```

Array[10][6][4] partition dimension 3

Array_0[10][6]
Array_1[10][6]
Array_2[10][6]
Array_3[10][6]

```
#pragma HLS array_partition variable=Array complete dim=1
```

Array[10][6][4] partition dimension 1

Array_0[6][4]
Array_1[6][4]
Array_2[6][4]
Array_3[6][4]
Array_4[6][4]
Array_5[6][4]
Array_6[6][4]
Array_7[6][4]
Array_8[6][4]
Array_9[6][4]

```
#pragma HLS array_partition variable=Array complete dim=0
```

Array[10][6][4] partition dimension 0 → **10 x 6 x 4 = 240 registers**

EXAMPLES: Array_Partition Complete

```
#include "example.h"

dout_t example(din_t mem[N]) {

#pragma HLS ARRAY_PARTITION variable=mem complete

    dout_t total = 0;

    for (int i = 2; i < N; ++i)
        total += mem[i] + mem[i - 1] + mem[i - 2];

    return total;

}
```

```
Timing:
  * Summary:
  +--------+----------+----------+------------+
  |  Clock |  Target  | Estimated| Uncertainty|
  +--------+----------+----------+------------+
  |ap_clk  | 25.00 ns | 4.339 ns |   3.12 ns  |
  +--------+----------+----------+------------+

Latency:
  * Summary:
  +---------+---------+-----------+-----------+-----+-----+---------+
  |  Latency (cycles) |  Latency (absolute)   |  Interval | Pipeline|
  |   min   |   max   |    min    |    max    | min | max |   Type  |
  +---------+---------+-----------+-----------+-----+-----+---------+
  |       63|       63| 1.575 us  | 1.575 us  |   63|   63|   none  |
  +---------+---------+-----------+-----------+-----+-----+---------+
```

```
* Loop:
+----------+---------+---------+----------+----------+----------+-----+---------+
|          |  Latency (cycles) | Iteration|  Initiation Interval  | Trip |         |
| Loop Name|   min   |   max   |  Latency |  achieved |   target  | Count| Pipelined|
+----------+---------+---------+----------+----------+----------+-----+---------+
|- Loop 1  |       62|       62|         1|         -|         -|   62|    no   |
+----------+---------+---------+----------+----------+----------+-----+---------+
```

```
* Summary:
+-----------------+---------+-------+--------+--------+-----+
|       Name      | BRAM_18K| DSP48E|    FF  |   LUT  | URAM|
+-----------------+---------+-------+--------+--------+-----+
|DSP              |        -|      -|       -|       -|    -|
|Expression       |        -|      -|       0|      76|    -|
|FIFO             |        -|      -|       -|       -|    -|
|Instance         |        -|      -|       0|     273|    -|
|Memory           |        -|      -|       -|       -|    -|
|Multiplexer      |        -|      -|       -|     571|    -|
|Register         |        -|      -|      19|       -|    -|
+-----------------+---------+-------+--------+--------+-----+
|Total            |        0|      0|      19|     920|    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|
+-----------------+---------+-------+--------+--------+-----+
```

EXAMPLES: Array_Partition Block

```
#include "example.h"

dout_t example(din_t mem[N]) {

#pragma HLS ARRAY_PARTITION variable=mem block factor=4

    dout_t total = 0;

    for (int i = 2; i < N; ++i)
        total += mem[i] + mem[i - 1] + mem[i - 2];

    return total;

}
```

+ Timing:

* Summary:

Clock	Target	Estimated	Uncertainty
ap_clk	25.00 ns	3.142 ns	3.12 ns

+ Latency:

* Summary:

Latency (cycles)		Latency (absolute)		Interval		Pipeline
min	max	min	max	min	max	Type
187	187	4.675 us	4.675 us	187	187	none

* Loop:

Loop Name	Latency (cycles) min	Latency (cycles) max	Iteration Latency	Initiation Interval achieved	Initiation Interval target	Trip Count	Pipelined
- Loop 1	186	186	3	-	-	62	no

* Summary:

Name	BRAM_18K	DSP48E	FF	LUT	URAM
DSP	-	-	-	-	-
Expression	-	-	0	128	-
FIFO	-	-	-	-	-
Instance	-	-	0	63	-
Memory	-	-	-	-	-
Multiplexer	-	-	-	105	-
Register	-	-	42	-	-
Total	0	0	42	296	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

EXAMPLES: Array_Partition Cyclic

```
#include "example.h"

dout_t example(din_t mem[N]) {

#pragma HLS ARRAY_PARTITION variable=mem cyclic factor=4

    dout_t total = 0;

    for (int i = 2; i < N; ++i)
        total += mem[i] + mem[i - 1] + mem[i - 2];

    return total;

}
```

```
Timing:
  * Summary:
  +--------+----------+----------+------------+
  |  Clock |  Target  | Estimated| Uncertainty|
  +--------+----------+----------+------------+
  |ap_clk  | 25.00 ns | 3.998 ns |   3.12 ns  |
  +--------+----------+----------+------------+

Latency:
  * Summary:
  +---------+---------+----------+----------+-----+-----+---------+
  |  Latency (cycles) |  Latency (absolute) |  Interval | Pipeline|
  |   min   |   max   |    min   |    max   | min | max |   Type  |
  +---------+---------+----------+----------+-----+-----+---------+
  |      187|      187| 4.675 us | 4.675 us |  187|  187|   none  |
  +---------+---------+----------+----------+-----+-----+---------+
```

```
* Loop:
+----------+---------+---------+----------+----------+----------+------+----------+
|          |  Latency (cycles) | Iteration|  Initiation Interval  | Trip |          |
| Loop Name|   min   |   max   |  Latency | achieved |   target | Count| Pipelined|
+----------+---------+---------+----------+----------+----------+------+----------+
|- Loop 1  |      186|      186|         3|         -|         -|    62|    no    |
+----------+---------+---------+----------+----------+----------+------+----------+
```

```
* Summary:
+--------------------+---------+-------+---------+---------+-----+
|        Name        | BRAM_18K| DSP48E|    FF   |   LUT   | URAM|
+--------------------+---------+-------+---------+---------+-----+
|DSP                 |        -|      -|        -|        -|    -|
|Expression          |        -|      -|        0|      116|    -|
|FIFO                |        -|      -|        -|        -|    -|
|Instance            |        -|      -|        0|       63|    -|
|Memory              |        -|      -|        -|        -|    -|
|Multiplexer         |        -|      -|        -|      105|    -|
|Register            |        -|      -|       37|        -|    -|
+--------------------+---------+-------+---------+---------+-----+
|Total               |        0|      0|       37|      284|    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|
+--------------------+---------+-------+---------+---------+-----+
```

A Comparison

	Default	Restructured	Pragma HLS Partition		
			Complete	Block factor = 4	Cyclic factor = 4
Estimated Clock (ns)	3.390	2.534	4.339	3.142	3.998
Latency (cycle)	187	126	63	187	187
Latency (μs)	4.675	3.150	1.575	4.675	4.675
Resources (FF)	35	50	19	42	37
Resources (LUT)	164	152	920	296	284
Resource (Others)	0	0	0	0	0

TAC-HEP 2026

#pragma HLS array_rehape

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

Pragma HLS array_reshape

#pragma HLS array_reshape variable=<name> <type> factor=<int> dim=<int>

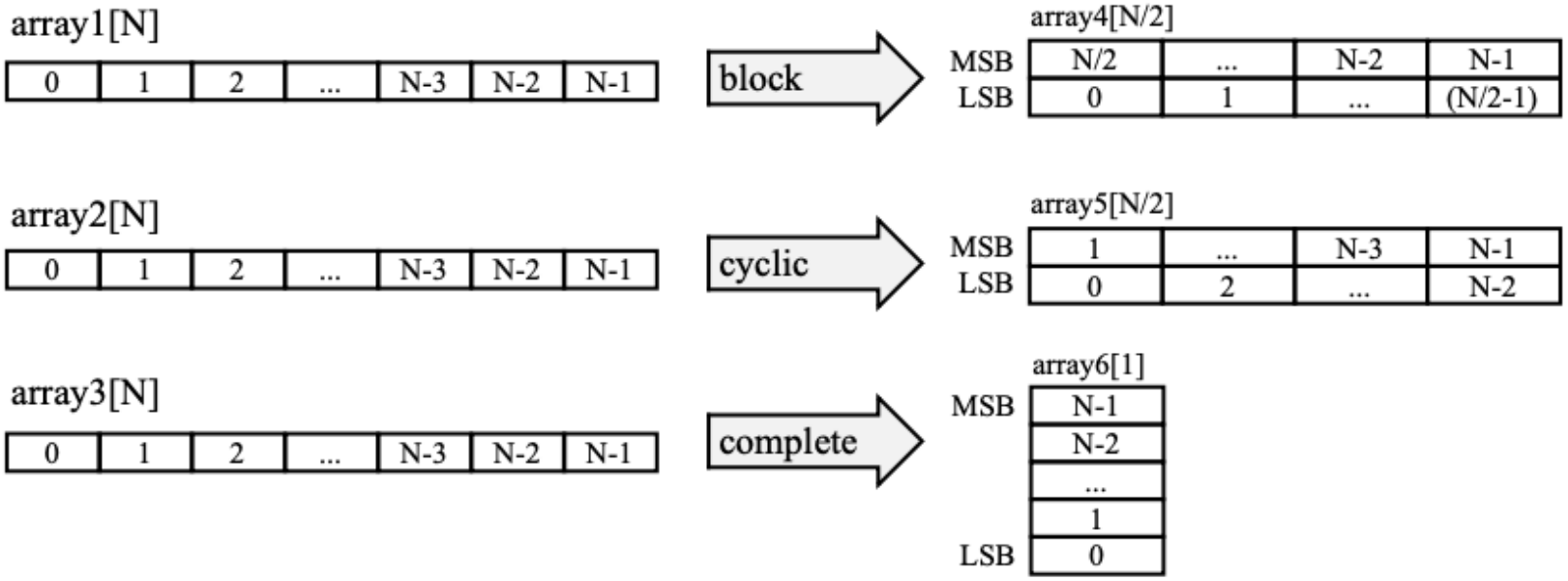
- ARRAY_RESHAPE pragma reforms the array with vertical remapping and concatenating elements of arrays by increasing bit-widths
- Reduces the number of block RAM consumed while providing parallel access to the data
- Pragma creates a new array with fewer elements but with greater bit-width, allowing more data to be accessed in a single clock cycle

Pragma HLS array_reshape

#pragma HLS array_reshape variable=<name> <type> factor=<int> dim=<int>

Cyclic: Cyclic partitioning creates smaller arrays by interleaving elements from the original array
Block: Block partitioning creates smaller arrays from consecutive N-blocks of the original array
Complete: Complete partitioning decomposes the array into individual elements

- For a 1-D array, this corresponds to resolving a memory into individual registers



Pragma HLS array_reshape: Complete

#pragma HLS array_reshape variable=<name> <type> factor=<int> dim=<int>

```
#include "example.h"

dout_t example(din_t mem[N]) {

#pragma HLS ARRAY_RESHAPE variable=mem complete

   dout_t total = 0;

   for (int i = 2; i < N; ++i)
      total += mem[i] + mem[i - 1] + mem[i - 2];

   return total;
}
```

```
Timing:
  * Summary:
  +--------+----------+----------+------------+
  |  Clock |  Target  | Estimated| Uncertainty|
  +--------+----------+----------+------------+
  |ap_clk  | 25.00 ns | 5.988 ns |   3.12 ns  |
  +--------+----------+----------+------------+

Latency:
  * Summary:
  +---------+---------+----------+----------+-----+-----+---------+
  |  Latency (cycles) |  Latency (absolute) |  Interval | Pipeline|
  |   min   |   max   |    min   |    max   | min | max |   Type  |
  +---------+---------+----------+----------+-----+-----+---------+
  |       63|       63| 1.575 us | 1.575 us |   63|   63|   none  |
  +---------+---------+----------+----------+-----+-----+---------+
```

```
* Loop:
+----------+---------+---------+----------+----------+----------+------+----------+
|          |  Latency (cycles) | Iteration|  Initiation Interval  | Trip |          |
| Loop Name|   min   |   max   |  Latency |  achieved |   target  | Count| Pipelined|
+----------+---------+---------+----------+----------+----------+------+----------+
|- Loop 1  |       62|       62|         1|         -|         -|    62|    no    |
+----------+---------+---------+----------+----------+----------+------+----------+
```

```
+---------------------+---------+-------+---------+---------+-----+
|         Name        | BRAM_18K| DSP48E|    FF   |   LUT   | URAM|
+---------------------+---------+-------+---------+---------+-----+
|DSP                  |        -|      -|        -|        -|    -|
|Expression           |        -|      -|        0|     5897|    -|
|FIFO                 |        -|      -|        -|        -|    -|
|Instance             |        -|      -|        -|        -|    -|
|Memory               |        -|      -|        -|        -|    -|
|Multiplexer          |        -|      -|        -|       33|    -|
|Register             |        -|      -|       19|        -|    -|
+---------------------+---------+-------+---------+---------+-----+
|Total                |        0|      0|       19|     5930|    0|
+---------------------+---------+-------+---------+---------+-----+
|Available SLR        |     1440|   2280|   788160|   394080|  320|
+---------------------+---------+-------+---------+---------+-----+
|Utilization SLR (%)  |        0|      0|    ~0   |        1|    0|
+---------------------+---------+-------+---------+---------+-----+
|Available            |     4320|   6840|  2364480|  1182240|  960|
+---------------------+---------+-------+---------+---------+-----+
|Utilization (%)      |        0|      0|    ~0   |    ~0   |    0|
+---------------------+---------+-------+---------+---------+-----+
```

TAC-HEP 2026

array_partition vs array_reshape

PARTITION vs RESHAPE

	Pragma HLS PARTITION			Pragma HLS RESHAPE		
	Complete	Block factor = 4	Cyclic factor = 4	Complete	Block factor = 4	Cyclic factor = 4
Estimated Clock (ns)	4.339	3.142	3.998	5.988	2.969	3.117
Latency (cycle)	63	187	187	63	187	187
Latency (μs)	1.575	4.675	4.675	1.575	4.675	4.675
Resources (FF)	19	42	37	19	46	42
Resources (LUT)	920	296	284	5930	488	476
Resource (Others)	0	0	0	0	0	0

PARTITION vs RESHAPE: Expression

PARTITION

* Expression:

Variable Name	Operation	DSP48E	FF	LUT	Bitwidth P0	Bitwidth P1
i_fu_1056_p2	+	0	0	15	7	1
ret_V_1_fu_1040_p2	+	0	0	16	9	9
ret_V_fu_1026_p2	+	0	0	15	8	8
total_V_fu_1050_p2	+	0	0	17	10	10
icmp_ln9_fu_938_p2	icmp	0	0	11	7	8
ap_condition_486	or	0	0	2	1	1
Total		0	0	76	42	37

* Multiplexer:

Name	LUT	Input Size	Bits	Total Bits
agg_result_V_0_reg_656	9	2	10	20
ap_NS_fsm	15	3	1	3
ap_phi_mux_phi_ln215_2_phi_fu_811_p124	269	63	7	441
ap_phi_mux_phi_ln215_phi_fu_681_p124	269	63	7	441
i_0_reg_667	9	2	7	14
Total	571	133	32	919

RESHAPE

* Expression:

Variable Name	Operation	DSP48E	FF	LUT	Bitwidth P0	Bitwidth P1
add_ln215_1_fu_154_p2	+	0	0	15	3	6
add_ln215_fu_103_p2	+	0	0	15	2	6
i_fu_211_p2	+	0	0	15	1	7
ret_V_1_fu_195_p2	+	0	0	16	9	9
ret_V_fu_144_p2	+	0	0	15	8	8
total_V_fu_205_p2	+	0	0	17	10	10
sub_ln215_1_fu_121_p2	-	0	0	16	9	9
sub_ln215_2_fu_172_p2	-	0	0	16	9	9
sub_ln215_fu_80_p2	-	0	0	16	9	9
icmp_ln9_fu_58_p2	icmp	0	0	11	7	8
lshr_ln215_1_fu_131_p2	lshr	0	0	1915	448	448
lshr_ln215_2_fu_182_p2	lshr	0	0	1915	448	448
lshr_ln215_fu_90_p2	lshr	0	0	1915	448	448
Total		0	0	5897	1411	1425

* Multiplexer:

Name	LUT	Input Size	Bits	Total Bits
agg_result_V_0_reg_36	9	2	10	20
ap_NS_fsm	15	3	1	3
i_0_reg_47	9	2	7	14
Total	33	7	18	37

PARTITION vs RESHAPE: Interface

PARTITION

* Summary:

RTL Ports	Dir	Bits	Protocol	Source Object	C Type
ap_clk	in	1	ap_ctrl_hs	example	return value
ap_rst	in	1	ap_ctrl_hs	example	return value
ap_start	in	1	ap_ctrl_hs	example	return value
ap_done	out	1	ap_ctrl_hs	example	return value
ap_idle	out	1	ap_ctrl_hs	example	return value
ap_ready	out	1	ap_ctrl_hs	example	return value
ap_return	out	10	ap_ctrl_hs	example	return value
mem_0_V	in	7	ap_none	mem_0_V	pointer
mem_1_V	in	7	ap_none	mem_1_V	pointer
mem_2_V	in	7	ap_none	mem_2_V	pointer
mem_3_V	in	7	ap_none	mem_3_V	pointer
mem_4_V	in	7	ap_none	mem_4_V	pointer
mem_5_V	in	7	ap_none	mem_5_V	pointer
mem_6_V	in	7	ap_none	mem_6_V	pointer
mem_7_V	in	7	ap_none	mem_7_V	pointer
mem_8_V	in	7	ap_none	mem_8_V	pointer
mem_9_V	in	7	ap_none	mem_9_V	pointer
mem_10_V	in	7	ap_none	mem_10_V	pointer
mem_11_V	in	7	ap_none	mem_11_V	pointer
mem_12_V	in	7	ap_none	mem_12_V	pointer
mem_13_V	in	7	ap_none	mem_13_V	pointer
mem_14_V	in	7	ap_none	mem_14_V	pointer
mem_15_V	in	7	ap_none	mem_15_V	pointer
mem_16_V	in	7	ap_none	mem_16_V	pointer
mem_17_V	in	7	ap_none	mem_17_V	pointer
mem_18_V	in	7	ap_none	mem_18_V	pointer
mem_19_V	in	7	ap_none	mem_19_V	pointer
mem_20_V	in	7	ap_none	mem_20_V	pointer
mem_21_V	in	7	ap_none	mem_21_V	pointer
mem_22_V	in	7	ap_none	mem_22_V	pointer
mem_23_V	in	7	ap_none	mem_23_V	pointer
mem_24_V	in	7	ap_none	mem_24_V	pointer
mem_25_V	in	7	ap_none	mem_25_V	pointer
mem_26_V	in	7	ap_none	mem_26_V	pointer
mem_27_V	in	7	ap_none	mem_27_V	pointer
mem_28_V	in	7	ap_none	mem_28_V	pointer
mem_29_V	in	7	ap_none	mem_29_V	pointer
mem_30_V	in	7	ap_none	mem_30_V	pointer
mem_31_V	in	7	ap_none	mem_31_V	pointer
mem_32_V	in	7	ap_none	mem_32_V	pointer

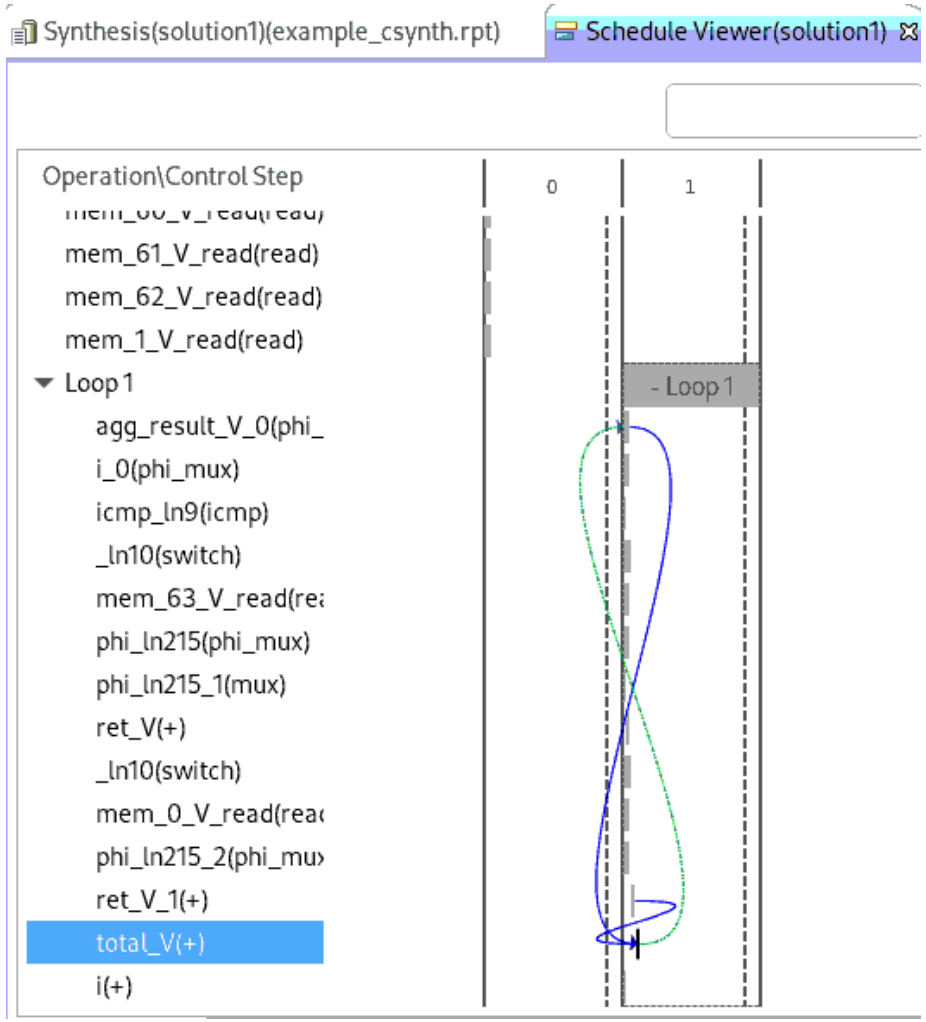
RESHAPE

* Summary:

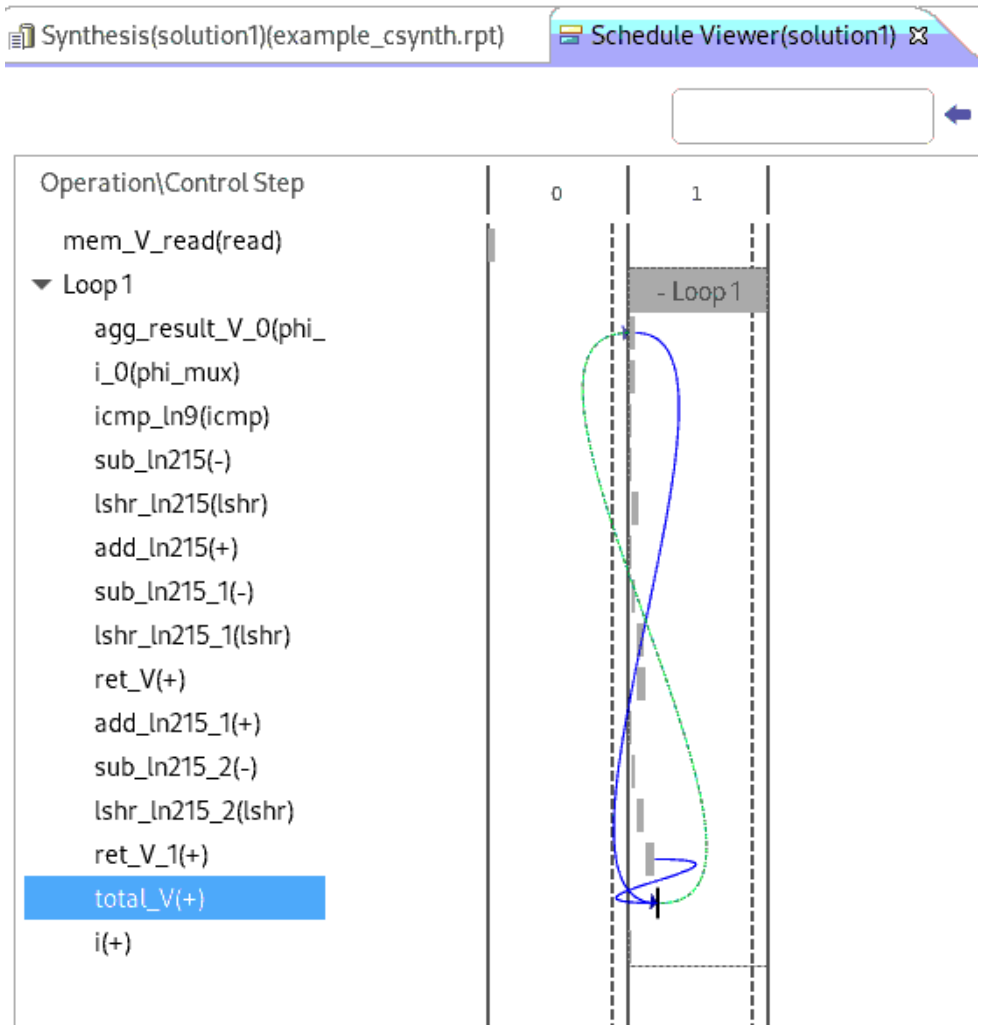
RTL Ports	Dir	Bits	Protocol	Source Object	C Type
ap_clk	in	1	ap_ctrl_hs	example	return value
ap_rst	in	1	ap_ctrl_hs	example	return value
ap_start	in	1	ap_ctrl_hs	example	return value
ap_done	out	1	ap_ctrl_hs	example	return value
ap_idle	out	1	ap_ctrl_hs	example	return value
ap_ready	out	1	ap_ctrl_hs	example	return value
ap_return	out	10	ap_ctrl_hs	example	return value
mem_V	in	448	ap_none	mem_V	pointer

PARTITION vs RESHAPE

PARTITION



RESHAPE



Pragma HLS array_reshape: Block

```
#include "example.h"

dout_t example(din_t mem[N]) {

#pragma HLS ARRAY_RESHAPE variable=mem block factor=4

   dout_t total = 0;

   for (int i = 2; i < N; ++i)
      total += mem[i] + mem[i - 1] + mem[i - 2];

   return total;
}
```

Timing:

* Summary:

Clock	Target	Estimated	Uncertainty
ap_clk	25.00 ns	2.969 ns	3.12 ns

Latency:

* Summary:

Latency (cycles) min	Latency (cycles) max	Latency (absolute) min	Latency (absolute) max	Interval min	Interval max	Pipeline Type
187	187	4.675 us	4.675 us	187	187	none

* Loop:

Loop Name	Latency (cycles) min	Latency (cycles) max	Iteration Latency	Initiation Interval achieved	Initiation Interval target	Trip Count	Pipelined
- Loop 1	186	186	3	-	-	62	no

* Summary:

Name	BRAM_18K	DSP48E	FF	LUT	URAM
DSP	-	-	-	-	-
Expression	-	-	0	428	-
FIFO	-	-	-	-	-
Instance	-	-	-	-	-
Memory	-	-	-	-	-
Multiplexer	-	-	-	60	-
Register	-	-	46	-	-
Total	0	0	46	488	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

Pragma HLS array_reshape: Cyclic

```
#include "example.h"

dout_t example(din_t mem[N]) {

#pragma HLS ARRAY_RESHAPE variable=mem cyclic factor=4

   dout_t total = 0;

   for (int i = 2; i < N; ++i)
      total += mem[i] + mem[i - 1] + mem[i - 2];

   return total;
}
```

```
Timing:
  * Summary:
  +--------+----------+----------+------------+
  |  Clock |  Target  | Estimated| Uncertainty|
  +--------+----------+----------+------------+
  |ap_clk  | 25.00 ns | 3.117 ns |   3.12 ns  |
  +--------+----------+----------+------------+

Latency:
  * Summary:
  +---------+---------+----------+----------+-----+-----+---------+
  |  Latency (cycles) |  Latency (absolute) |  Interval | Pipeline|
  |   min   |   max   |    min   |    max   | min | max |   Type  |
  +---------+---------+----------+----------+-----+-----+---------+
  |      187|      187| 4.675 us | 4.675 us |  187|  187|   none  |
  +---------+---------+----------+----------+-----+-----+---------+
```

```
* Loop:
+----------+---------+---------+----------+----------+----------+-----+----------+
|          |  Latency (cycles) | Iteration|  Initiation Interval  | Trip |          |
| Loop Name|   min   |   max   |  Latency |  achieved |   target  | Count| Pipelined|
+----------+---------+---------+----------+----------+----------+-----+----------+
|- Loop 1  |      186|      186|         3|         -|         -|   62|    no    |
+----------+---------+---------+----------+----------+----------+-----+----------+
```

```
* Summary:
+--------------------+---------+-------+---------+---------+-----+
|        Name        | BRAM_18K| DSP48E|    FF   |   LUT   | URAM|
+--------------------+---------+-------+---------+---------+-----+
|DSP                 |        -|      -|        -|        -|    -|
|Expression          |        -|      -|        0|      416|    -|
|FIFO                |        -|      -|        -|        -|    -|
|Instance            |        -|      -|        -|        -|    -|
|Memory              |        -|      -|        -|        -|    -|
|Multiplexer         |        -|      -|        -|       60|    -|
|Register            |        -|      -|       42|        -|    -|
+--------------------+---------+-------+---------+---------+-----+
|Total               |        0|      0|       42|      476|    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|
+--------------------+---------+-------+---------+---------+-----+
```

Homework #3

- Use the program from Homework#1 (multiplication of two arrays/vectors and matrix with at least 10 elements.
  1. Include HLS Pragma Interface + HLS Pragma array_partition
     a. Partition both dimensions (valid for matrix only)
     b. Partition only one dimension
     c. Compare results from different partitioning

Share the results and your observations about different partitioning options used.

TAC-HEP 2026

Questions?

TAC-HEP 2026

Extra Slides

TAC-HEP 2026

HLS Setup on cmstrigger02

https://github.com/varuns23/TAC-HEP-FPGA/blob/main/hls-setup/readme.md

Setup

Setup Port Forwarding for RDP

- From your local machine, run:
  - `ssh -L 3389:localhost:3389 -J <username>@login.hep.wisc.edu <username>@cmstrigger02.hep.wisc.edu`
- Keep this terminal open - it maintains the RDP tunnel

Connect Using Remote Desktop (RDP Client)

- Use Microsoft's RDP client (called `Windows App`) available for macOS and Windows.
- **Setup:** Download & install the `Windows App`, Open the app and click the **+** icon, then select `Add PC`
- **Configure:** Enter the IP address: `localhost:3389` or `127.0.0.1:3389`
- **Connect:** Double-click the PC icon, enter your UW computing and , and click **Connect**
- You should now see the remote desktop of `cmstrigger02`

https://github.com/varuns23/TAC-HEP-FPGA/blob/main/hls-setup/readme.md

Setup workign Directory

After logging in via RDP, open a terminal inside the remote desktop

- `mkdir -p /scratch/`whoami`` (if not there already)
- `cd /scratch/`whoami``
  - **For Vitis HLS:**
    - `Source /opt/Xilinx/Vitis/2020.1/settings64.sh`
    - `cd /scratch/`whoami``
    - `vitis_hls`
  - **For Vivado HLS:**
    - `Source /opt/Xilinx/Vivado/2020.1/settings64.sh`
    - `cd /scratch/`whoami``
    - `vivado_hls`

Homework# 2 (lecture-6)

- Run following C++ example using vitis and share your conclusion.
  - https://github.com/varuns23/TAC-HEP-FPGA/blob/main/2026/lecture06/lex6-ex01.cpp

- Write a progam to sort 16 object and with Vitis

Homework#1 (lecture-5)

```
1   #include <iostream>
2   #define N 10
3
4   void matrix_add(int A[N][N], int B[N][N], int C[N][N]) {
5
6       for (int i = 0; i < N; i++) {
7           for (int j = 0; j < N; j++) {
8               C[i][j] = A[i][j] + B[i][j];
9           }
10      }
11  }
```

1. Run on HLS for the above example of matrix addition and check the resource utilization
2. Compare with a vector addition and matrix multilplication.

Share your resource utilization for all three cases and your observation about the same.

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

Terminology

- **HLS file:** C/C++ code that will be synthesised and run on FPGA
- **Test bench (TB) file:** C/C++ code that is run to test the HLS code. It calls the HLS functions and can run tests on their output, e.g. C asserts
- **Tcl scripts:** set of tcl instructions executed by the Vivado HLS shell

- **Synthesis:** C/C++ ➞ HDL lang (VHDL/Verilog)
- **Project:** Collection of HLS and test bench (TB) files
  - Has a top-level function name that is the starting point for synthesis
- **Solution:** specific implementation of project
  - Runs on a specific device at a specific clock frequency
- **C simulation:** HLS+TB files are compiled with gcc against HLS headers and lib and plainly run as any other executable
- **C/RTL cosimulation:** synthesized HLS code is run on simulator and results tested on the C/C++ test bench

TAC-HEP 2026

Pragma HLS Interface

Advanced eXtensible Interface (AXI)

AXI is part of ARM AMBA, a family of micro controller buses first introduced in 1996

AXI4 is for memory-mapped interfaces and allows high throughput bursts of up to 256 data transfer cycles with just a single address phase.

There are three types of AXI4 interfaces:

- **AXI4 (m_axi):** For high-performance memory-mapped requirements
  - Specify on array and pointers (and references in C++)

- **AXI4-Lite (s_axilite):** For simple, low-throughput memory-mapped communication (for example, to and from control and status registers)
  - Can be used on any type of argument except streams

- **AXI4-Stream (axis):** For high-speed streaming data
  - Use this protocol on input arguments or output arguments only,

Control Signal: 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.