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

FPGA module training

Week-8

Lecture-14: 18/03/2025

Varun Sharma

University of Wisconsin – Madison, USA

Content

- HLS Pragmas:
  - Array Partition
  - Array Reshape

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 2025

#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

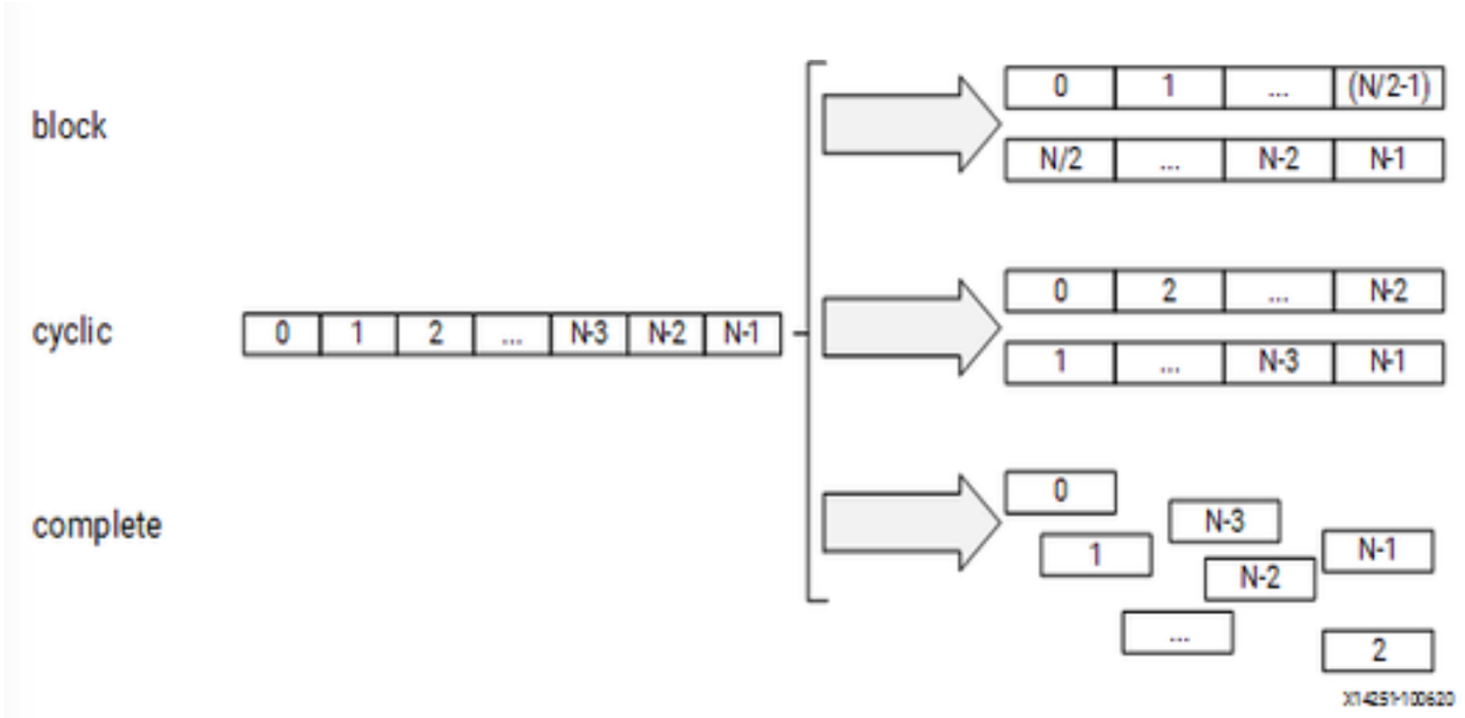
```
#pragma HLS array_partition variable=<name>  <type> factor=<int> dim=<int>
```

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)		Iteration	Initiation Interval		Trip	
	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	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

Assignment #5

- Write a HLS code to multiply two Matrices (2-dimension) with at least 10 elements.
  1. Include HLS Pragma Interface + HLS Pragma array_partition
     a. Partition both dimensions
     b. Partition only one dimension

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

Reminder: Assignments

- Assignment-1 (13-02-2025)
- Assignment-2 (18-02-2025)
- Assignment-3 (27-02-2025)
- Assignment-4 (18-03-2025)
- Assignment-5 (18-03-2025)

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

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

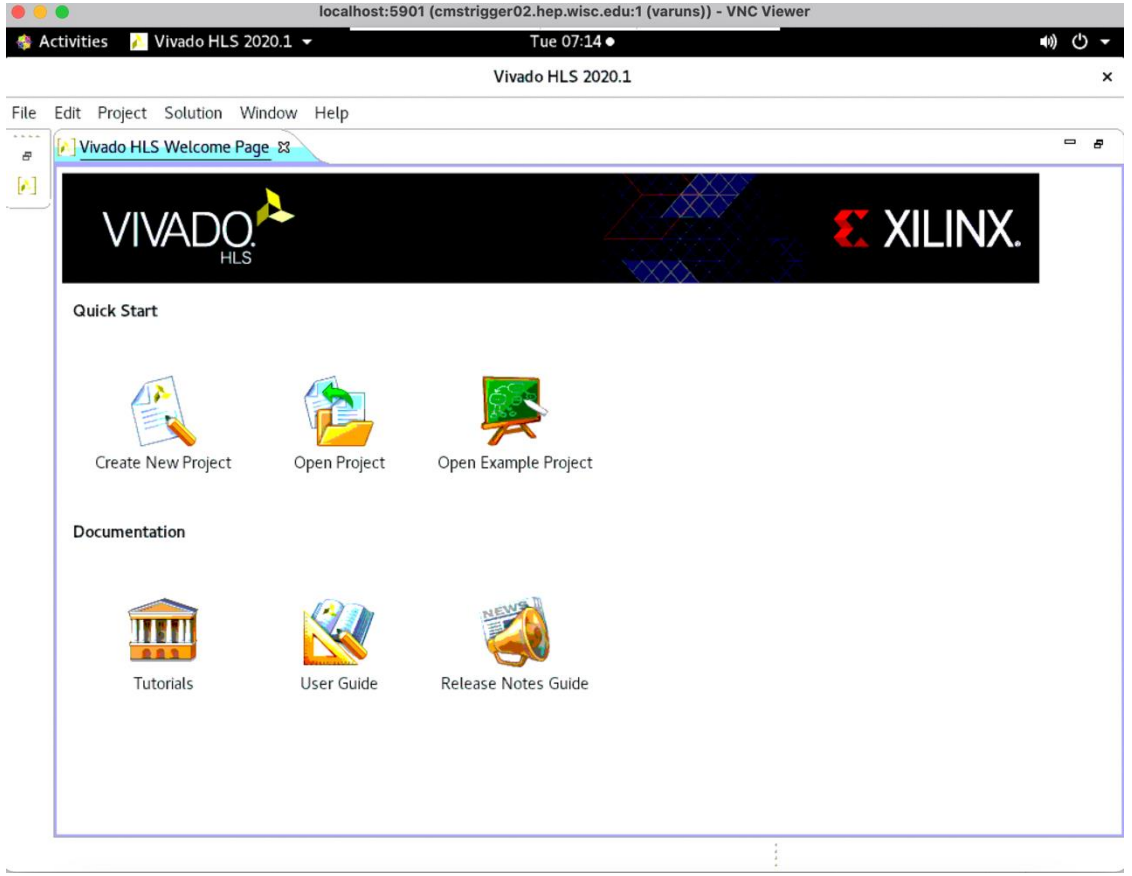
Reminder: HLS Setup

- 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