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

FPGA module training

Week-10

Lecture-19: 10/04/2025

Varun Sharma

University of Wisconsin – Madison, USA

Content

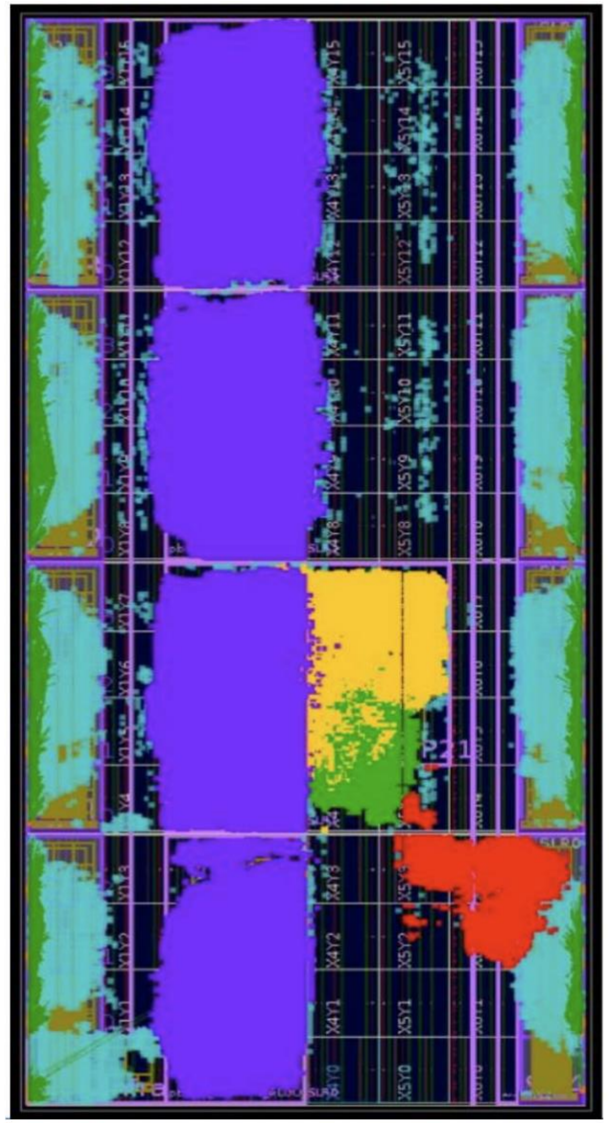
<u>So far</u>

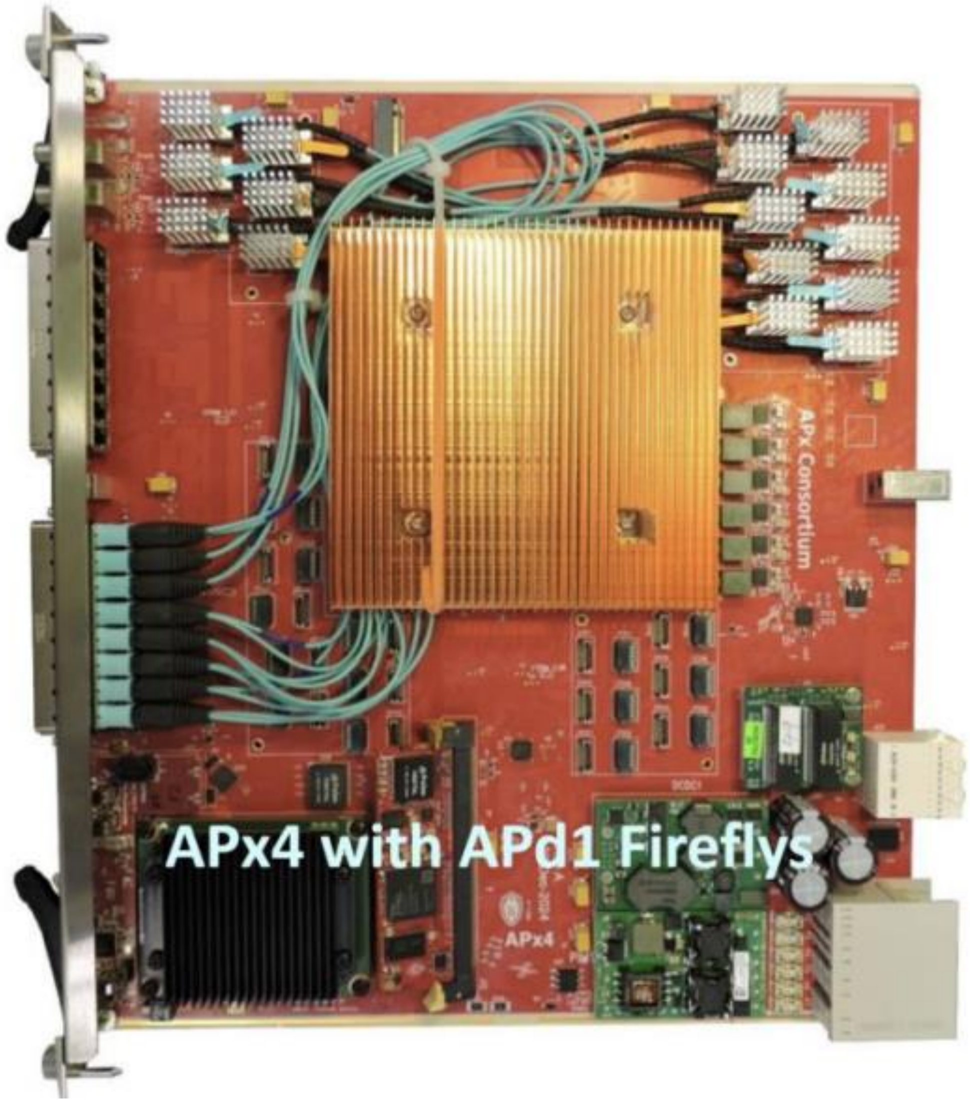
- HLS Pragmas:
  - Interface
  - Array Partition
  - Array reshape
  - Pipeline
  - Dataflow
  - Latency
  - Allocation
  - Stable
  - Inline
  - Unroll

<u>Today</u>

- Unsupported C/C++ constructs
- Matrix Multiplication
- Examples for HLS Pragmas:
  - Unroll

Different IPs on VU13P





TAC-HEP 2025

Unsupported C/C++ Constructs

C/C++ constructs

HLS compilers support many C/C++ constructs, but some are not synthesizable.

- Coding changes may be required for successful synthesis and implementation.

For a function to be synthesized:

- It must fully contain the design's functionality
- No system calls to the operating system are allowed
- All C/C++ constructs must have fixed or bounded sizes
- The constructs' implementation must be unambiguous

System Calls

System calls are not synthesizable because they interact with the operating system, which is not present in the hardware environment where the synthesized design runs

Vitis HLS ignores certain system calls like printf() and fprintf(stdout,) if they only display data and don't affect algorithm execution.

Most system calls (e.g., getc(), time(), sleep()) are not synthesizable and should be removed before synthesis

Vitis HLS defines the __SYNTHESIS__ macro during synthesis.

- This macro can be used to conditionally exclude non-synthesizable code from the design

```
void hier_func4(din_t A, din_t B, dout_t *C, dout_t *D)
{
  dint_t apb, amb;
  sumsub_func(&A,&B,&apb,&amb);

#ifndef __SYNTHESIS__
  FILE *fp1; // The following code is ignored for synthesis
  char filename[255];
  sprintf(filename,Out_apb_%03d.dat,apb);
  fp1=fopen(filename,w);
  fprintf(fp1, %d \n, apb);
  fclose(fp1);
#endif
  shift_func(&apb,&amb,C,D);

}
```

Dynamic Memory Usage

- **Memory allocation system calls** like `malloc()`, `alloc()`, and `free()` rely on OS-managed resources and runtime behavior
- Such calls **cannot be synthesized** and must be removed from the design code
- A hardware design must be **fully self-contained**, with all required resources explicitly defined
- **Dynamic memory operations** must be replaced with **equivalent fixed or bounded representations** for synthesis

Because the coding changes impact the functionality of the design, AMD does not recommend using the `__SYNTHESIS__` macro.

Example

```c
#include "malloc_removed.h"
#include <stdlib.h>
//#define NO_SYNTH

dout_t malloc_removed(din_t din[N], dsel_t
width)
{
#ifdef NO_SYNTH
  long long *out_accum = malloc
(sizeof(long long));
  int* array_local = malloc (64 *
sizeof(int));
#else
  long long _out_accum;
  long long *out_accum = &_out_accum;
  int _array_local[64];
  int* array_local = &_array_local[0];
#endif
```

```c
int i,j;
LOOP_SHIFT:for (i=0;i<N-1; i++){
if (i<width)
  *(array_local+i)=din[i];
else
  *(array_local[i])=din[i]>>2;
}

*out_accum=0;

LOOP_ACCUM:for (j=0;j<N-1; j++) {
   *out_accum += *(array_local+j);
}

return *out_accum;
}
```

Dynamic Memory Usage

1. Add the user-defined macro **NO_SYNTH** to the code and modify the code.
2. Enable macro **NO_SYNTH**, execute the C/C++ simulation, and save the results.
3. Disable the macro **NO_SYNTH**, and execute the C/C++ simulation to verify that the results are identical.
4. Perform synthesis with the user-defined macro disabled.

This methodology ensures that the updated code is validated with C/C++ simulation and that the identical code is then synthesized

Pointer Limitation

✗General pointer casting is not supported by Vitis HLS

```
int num = 10;
void *ptr = &num;  // Void pointer pointing to an integer
// Cast the void pointer to an integer pointer
int *intPtr = (int *)ptr;
```

✓ Pointer arrays are supported

- ✓Given they points to scalar or an array of scalars
- ✗Arrays of pointers can't point to additional pointers

✗Function pointers are not supported

```
int (*funcPtr)(int, int);
```

Recursive Functions

✗Recursive functions can't be synthesized (function that can perform multiple recursions)

```
unsigned foo (unsigned n)
{
   if (n == 0 || n == 1) return 1;
   return (foo(n-2) + foo(n-1));
}
```

✗Tail recursions are also not allowed (finite number of function calls)

```
unsigned foo (unsigned m, unsigned n)
{
   if (m == 0) return n;
   if (n == 0) return m;
   return foo(n, m%n);
}
```

Standard Template Libraries (STL)

- Many C++ STLs contain function recursion and use dynamic memory allocation
- These can **NOT** be synthesized by Vitis HLS

- **Solution:**
  - Create a local function with identical functionality that does not feature recursion, dynamic memory allocation, or dynamic creating and destruction of objects.

- **Example:** `std::vector`, `std::map`, `std::list`, `std::sort`

Undefined Behaviors

The C/C++ undefined behaviors is allowed but may lead to a different behavior in simulation and synthesis

```
for (int i=0; i< N; i++) {
    int val; //un-initialized value
    if (i == 0) val = 0;
    else if (cond) val = 1;
    // val may have intermediate value here
    A[i] = val; //undefined behavior
    val++;  // dead code
```

Behavior between GCC and HLS when compiling code is likely to be different

Lead to a mis-match during RTL/co-simulation

- In GCC compiled for CPU, the value of **val** may be retained across loop iterations, as it could remain in the same register or stack location

- **Good Practise:**
    - Initialize val at the start of each iteration if this behavior is expected.
    - Move the declaration of **val** above the loop so that its lifetime matches the intended reuse.

Do not expect the compiler to infer a specific defined RTL behavior from undefined C/C++ behavior

Some common errors/warnings

```
WARNING: [RTGEN 206-101] Setting dangling out port 'example/A_WEN_A' to 0 WARNING: [RTGEN 206-101] Setting dangling out port 'example/A_Din_A' to 0
```

This means HLS generated **write-enable (WEN) and data-in (Din) ports** for array A, **but they are never written to** in the design — so those outputs are dangling (unused) and set to 0.

```
#pragma HLS INTERFACE ap_const port=A
#pragma HLS INTERFACE ap_const port=B
```

These are **read-only and** won't generate write ports (no WEN, Din)

```
ERROR: [XFORM 203-801] Interface parameter bitwidth 'A.V' (example.cpp:8:1) must be a multiple of 8 for AXI4 master port
```

AXI4 memory-mapped interfaces **require data widths in bytes (multiples of 8 bits)**

Some common errors/warnings

```
WARNING: [SCHED 204-69] Unable to schedule 'load' operation ('A_load_2', example.cpp:22) on array 'A' due to limited memory ports.
Please consider using a memory core with more ports or partitioning the array 'A'.
```

This **warning** is super common in HLS when multiple accesses happen to the **same memory** (like arrays A, B, or C) in the **same clock cycle**, but the default memory core only has **one read and one write port**

- HLS maps arrays to **block RAMs** (usually single-port or dual-port).
- When you **pipeline loops** (like with #pragma HLS PIPELINE), multiple operations might try to **read/write to the same array at once**.
- Since **BRAM has limited ports**, it throws a scheduling warning

Try **partitioning array**: May get rid of the warning

TAC-HEP 2025

Matrix Multiplication

Matrix multiplication

```
#include <ap_int.h>
#include <hls_stream.h>

#include "example.h"

// Top-level function for HLS
void example(din_t A[N][N], din_t B[N][N], din_t C[N][N]) {
#pragma HLS INTERFACE m_axi port=A
#pragma HLS INTERFACE m_axi port=B
#pragma HLS INTERFACE m_axi port=C

    // Matrix multiplication
    for (size_t i = 0; i < N; i++) {
        for (size_t j = 0; j < N; j++) {
#pragma HLS PIPELINE II=1
            din_t sum = 0;
            for (size_t k = 0; k < N; k++) {

                sum += A[i][k] * B[k][j];
            }
            C[i][j] = sum;
        }
    }
}
```

TAC-HEP 2025

Pragmas Unroll

UNROLL Example

```
#include <ap_int.h>
#include <hls_stream.h>

#include "example.h"

void example(din_t A[N], din_t B[N], din_t C[N]) {

 for (size_t i = 0; i < N; ++i) {
#pragma HLS UNROLL factor = 4
    C[i] = A[i] + B[i];
  }
}
```

Assignment-6

- Use example in slide-3 of lecture 17 to reduce resource utilization – specially the DSP usage (https://github.com/varuns23/TAC-HEP-FPGA/tree/main/tutorial/wk9lec17/ex-func)
  - You can use a combination of sub-set of following pragmas:
    - Array Partition
    - Array reshape
    - Pipeline
    - Dataflow
    - Latency
    - Allocation
    - INLINE
  - **Objective: To have DSP usage less than 10**
- Refer to ex-all folder for example with pragmas

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**

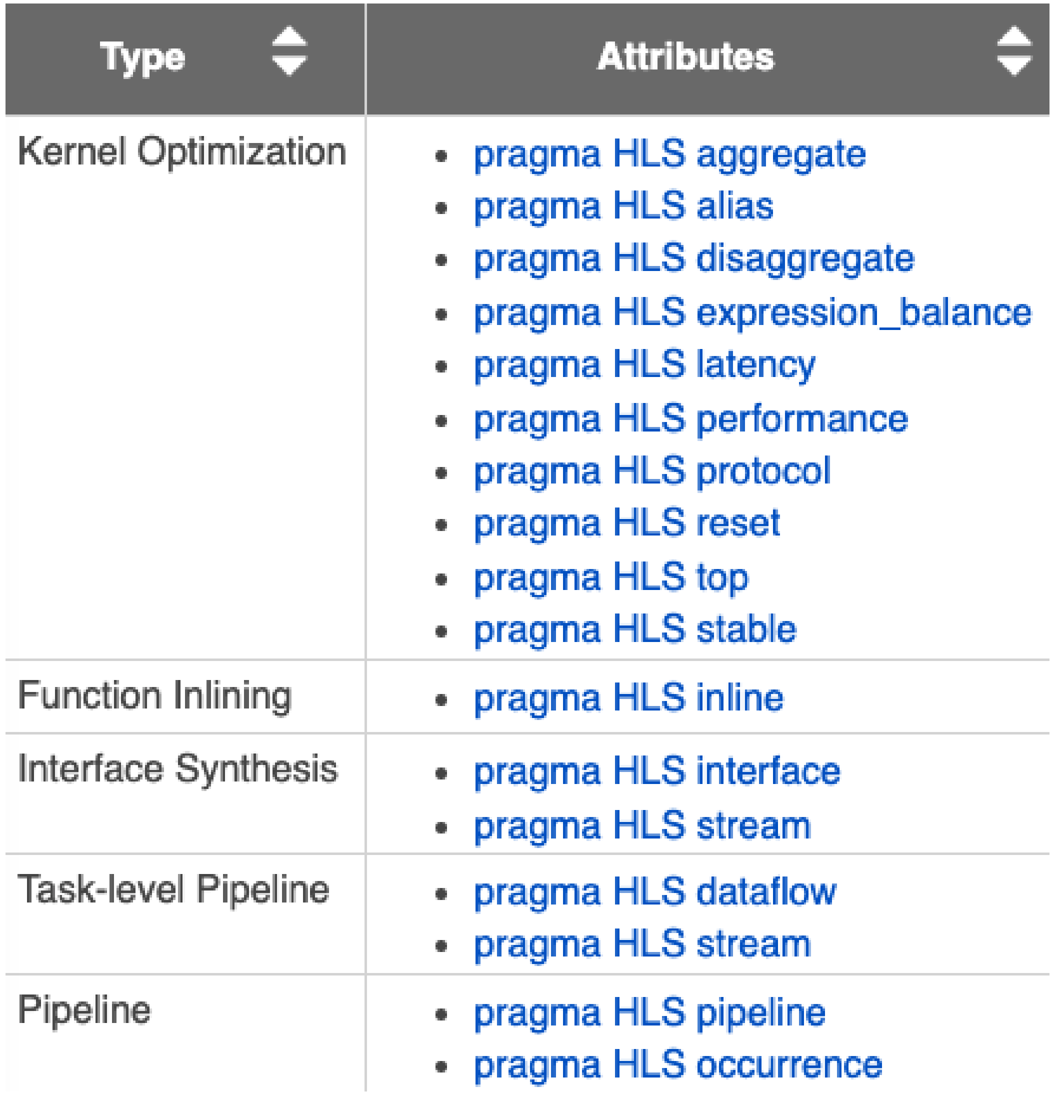
Submit in 2 weeks from date of assignment

TAC-HEP 2025

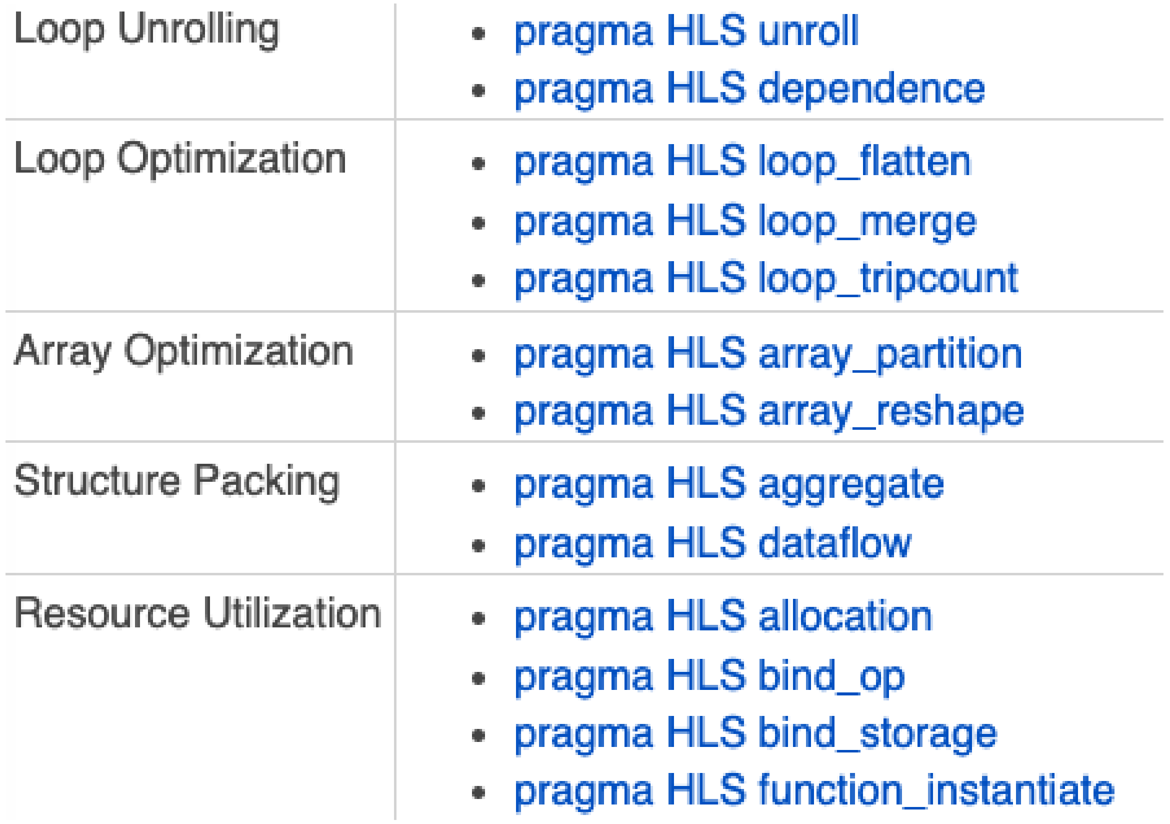
Questions?

Acknowledgements:
- https://docs.amd.com/r/2024.1-English/ug1399-vitis-hls
- 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

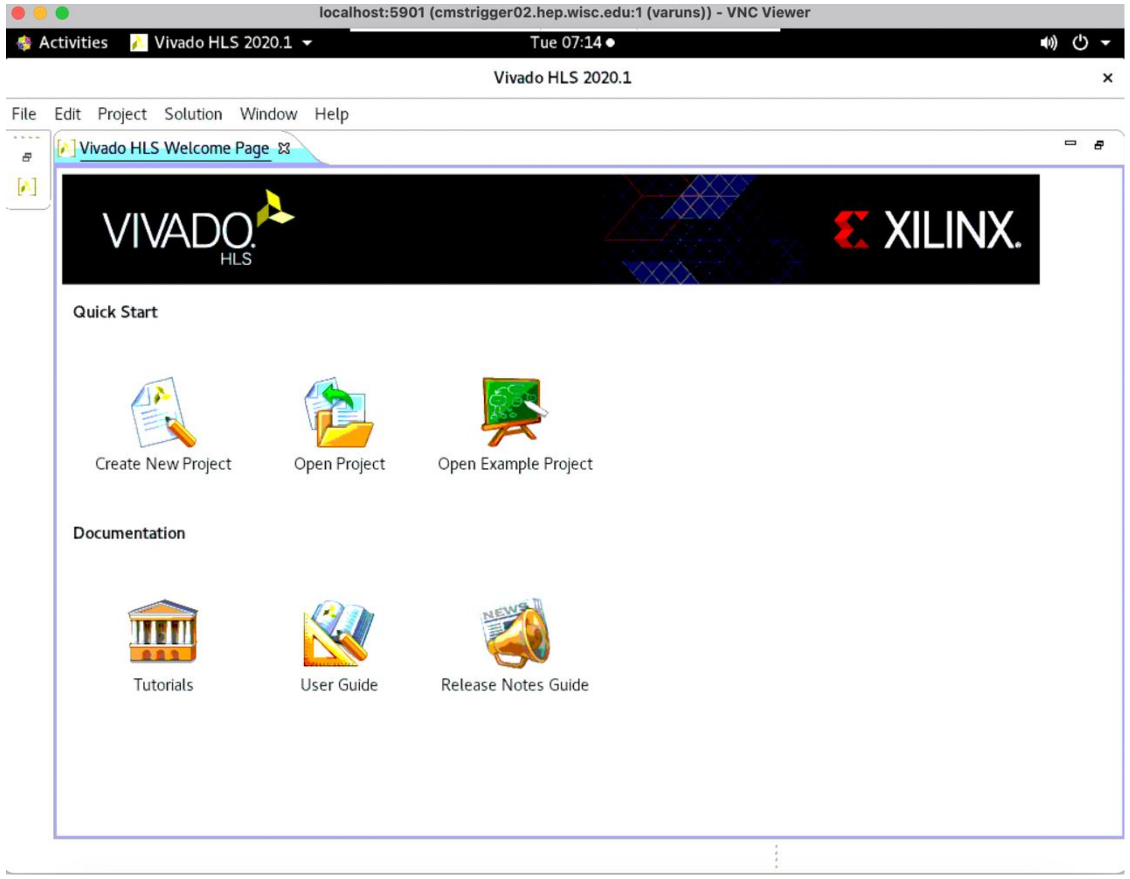
Reminder: HLS Setup

- 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