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

FPGA module training

Week-9

Lecture-16: 25/03/2025

Varun Sharma

University of Wisconsin – Madison, USA

UNIVERSITY OF WISCONSIN–MADISON

Content

So far

- HLS Pragmas:
  - Interface
  - Array Partition
  - Array reshape
  - Pipeline

Today

- HLS Pragmas:
  - Dataflow
  - Latency
  - Allocation

TAC-HEP 2025

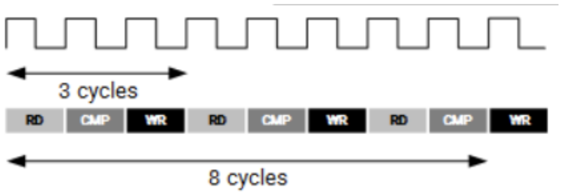
#pragma HLS Pipeline

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

Pragma HLS Pipeline

#pragma HLS pipeline II=<int>

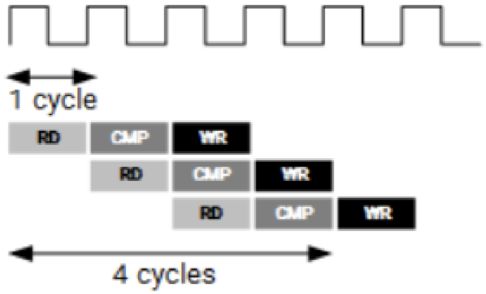
- The PIPELINE pragma reduces the initiation interval (II) for a function or loop by allowing the concurrent execution of operations
- A pipelined function or loop can process new inputs every <N> clock cycles
- If HLS can't create a design with the specified II, it issues a warning and creates a design with the lowest possible II



(A) Without Loop Pipelining

Without Loop pipelining

```
void func(input, output){
...
  for(i=0; i>=N; i++){
#pragma HLS pipeline II=2
    op_read;
    op_compute;
    op_write;
  }
...
}
```



With Loop pipelining

#pragma HLS Dataflow

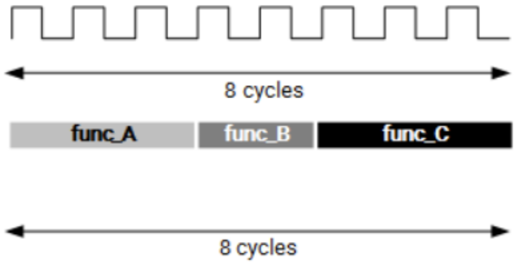
https://docs.amd.com/r/en-US/ug1399-vitis-hls/pragma-HLS-dataflow

Pragma HLS Dataflow

Task-level pipeline

#pragma HLS dataflow

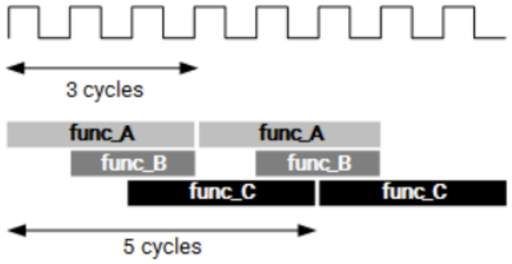
- **Enables task-level pipelining:** allow functions and loops to overlap in their operation
  - Increases the concurrency of the RTL implementation & thus the overall throughput of the design
- In the absence of any directives that limit resources (like pragma HLS allocation), HLS seeks to minimize latency & improve concurrency
  - Data dependencies can limit this, hence proper dataflow is needed



Without DATAFLOW pipelining

```
void top(a, b, c, d){
   ...
   func_A(a,b,i1);
   func_B(c,i1,i2);
   func_C(i2,d);

   ...
   return d;
}
```



With DATAFLOW pipelining

Pragma HLS Dataflow

Task-level pipeline

#pragma HLS dataflow

- **Enables task-level pipelining:** allow functions and loops to overlap in their operation
  - Increases the concurrency of the RTL implementation & thus the overall throughput of the design
- In the absence of any directives that limit resources (like pragma HLS allocation), HLS seeks to minimize latency & improve concurrency
  - Data dependencies can limit this, hence proper dataflow is needed

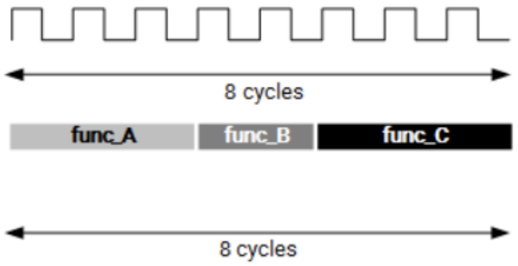
Example:

- Functions/loops that access arrays must finish all read/write accesses to the arrays before they complete
- Prevent the next function or loop that consumes the data from starting operation
- The DATAFLOW optimization enables the operations in a function or loop to start operation before the previous function or loop completes all its operations

Pragma HLS Dataflow

Task-level pipeline

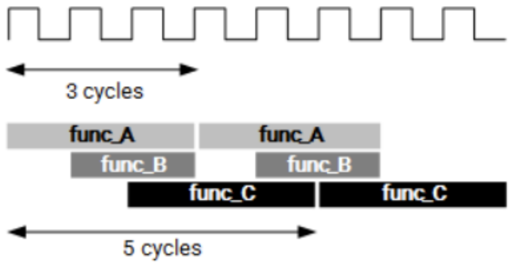
#pragma HLS dataflow



Without DATAFLOW pipelining

```
void top(a, b, c, d){
   ...
   func_A(a,b,i1);
   func_B(c,i1,i2);
   func_C(i2,d);

   ...
   return d;
}
```



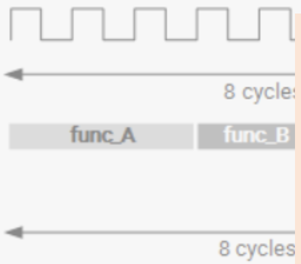
With DATAFLOW pipelining

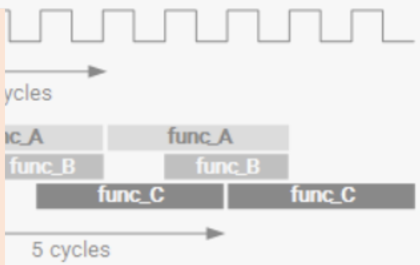
- ✗ Bypassing tasks
- ✗ Feedback between tasks
- ✗ Conditional execution of tasks
- ✗ Loops with multiple exit conditions

Pragma HLS Dataflow - Example

Task-level pipeline

#pragma HLS dataflow





Without DATAFL

ATAFLOW pipelining

- ✗ Bypassing tasks
- ✗ Feedback between tasks
- ✗ Conditional execution of tasks
- ✗ Loops with multiple exit conditions

For t

next

- ✓ HLS tool issues a message and does not perform DATAFLOW optimization
- ✓ Use the STABLE pragma to mark variables within DATAFLOW regions to be stable to avoid concurrent read or write of variables.
- ✓ No hierarchial implementation

Pragma HLS Dataflow - Example

```
#include "example.h"

void example (
  unsigned int in[N],
  short a,
  short b,
  unsigned int c,
  unsigned int out[N]
  ) {

   unsigned int x, y;
   unsigned int tmp1, tmp2, tmp3;

for_Loop: for (unsigned int i=0 ; i < N; i++) {

        x = in[i];
        tmp1 = func(1, 2);
        tmp2 = func(2, 3);
        tmp3 = func(1, 4);

        y = a*x + b + squared(c) + tmp1 + tmp2 + tmp3;

        out[i] = y;
      }
}

unsigned int squared(unsigned int a)
{
  unsigned int res = 0;
  res = a*a;
  return res;
}

unsigned int func(short a, short b){

  unsigned int res;
  res= a*a;
  res= res*b*a;
  res= res + 3;

  return res;
}
```

#pragma HLS dataflow

```
void example (
  unsigned int in[N],
  short a,
  short b,
  unsigned int c,
  unsigned int out[N]
  ) {

   unsigned int x, y;
   unsigned int tmp1, tmp2, tmp3;

#pragma HLS dataflow

for_Loop: for (unsigned int i=0 ; i < N; i++) {
#pragma HLS Pipeline

        x = in[i];
        tmp1 = func(1, 2);
        tmp2 = func(2, 3);
        tmp3 = func(1, 4);

        y = a*x + b + squared(c) + tmp1 + tmp2 + tmp3;

        out[i] = y;
      }
}

unsigned int squared(unsigned int a)
{
  unsigned int res = 0;
  res = a*a;
  return res;
}

unsigned int func(short a, short b){

  unsigned int res;
  res= a*a;
  res= res*b*a;
  res= res + 3;

  return res;
}
```

Pragma HLS Dataflow

#pragma HLS dataflow

Without DATAFLOW pipelining

```
Timing:
  * Summary:
  +--------+----------+----------+------------+
  |  Clock |  Target  | Estimated| Uncertainty|
  +--------+----------+----------+------------+
  |ap_clk  | 25.00 ns | 7.401 ns |   3.12 ns  |
  +--------+----------+----------+------------+

Latency:
  * Summary:
  +---------+---------+----------+----------+-----+-----+---------+
  |  Latency (cycles) |  Latency (absolute) |  Interval | Pipeline|
  |   min   |   max   |    min   |    max   | min | max |   Type  |
  +---------+---------+----------+----------+-----+-----+---------+
  |      121|      121| 3.025 us | 3.025 us |  121|  121|   none  |
  +---------+---------+----------+----------+-----+-----+---------+
```

With DATAFLOW pipelining

```
Timing:
  * Summary:
  +--------+----------+----------+------------+
  |  Clock |  Target  | Estimated| Uncertainty|
  +--------+----------+----------+------------+
  |ap_clk  | 25.00 ns | 7.401 ns |   3.12 ns  |
  +--------+----------+----------+------------+

Latency:
  * Summary:
  +---------+---------+----------+----------+-----+-----+----------+
  |  Latency (cycles) |  Latency (absolute) |  Interval | Pipeline |
  |   min   |   max   |    min   |    max   | min | max |   Type   |
  +---------+---------+----------+----------+-----+-----+----------+
  |       62|       62| 1.550 us | 1.550 us |   63|   63| dataflow |
  +---------+---------+----------+----------+-----+-----+----------+
```

Pragma HLS Dataflow

#pragma HLS dataflow

Without DATAFLOW pipelining

== Utilization Estimates

* Summary:

Name	BRAM_18K	DSP48E	FF	LUT	URAM
DSP	-	-	-	-	-
Expression	-	5	0	154	-
FIFO	-	-	-	-	-
Instance	-	-	-	-	-
Memory	-	-	-	-	-
Multiplexer	-	-	-	30	-
Register	-	-	117	-	-
Total	0	5	117	184	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

With DATAFLOW pipelining

== Utilization Estimates

* Summary:

Name	BRAM_18K	DSP48E	FF	LUT	URAM
DSP	-	-	-	-	-
Expression	-	-	-	-	-
FIFO	-	-	-	-	-
Instance	-	5	115	214	-
Memory	-	-	-	-	-
Multiplexer	-	-	-	-	-
Register	-	-	-	-	-
Total	0	5	115	214	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 allocation

- Specifies instance restrictions to limit resource allocation in the implemented kernel
- Defines & can limit the number of RTL instances and hardware resources used to implement specific functions, loops, operations or cores

- Example: c-source code has 4 instances of a function *my_func*
  - ALLOCATION pragma can ensure that there is only one instance of of *my_func*
  - All 4 instances are implemented using the same RTL block
    - Reduces resource used by function but may impact performance

- **Operations:** additions, multiplications, array reads, & writes can be limited by ALLOCATION pragma

Pragma HLS allocation - Syntax

#pragma HLS allocation instances=<list> limit=<value> <type>

- **Instance<list>*:** Name of the function, operator, or cores
- **limit=<value>*:** Specifies the limit of instances to be used in kernel
- **<type>*:** Specifies the allocation applies to a function, an operator or a core (hardware component) used to create the design (such as adder, multiplier, BRAM)
  - *Function*: allocation applies to the functions listed in the instances=
  - *Operation*: applies to the operations listed in the instances=
  - *Core*: applies to the cores

Pragma HLS allocation - Example

#pragma HLS allocation instances=<list> limit=<value> <type>

Example1: Limits the number of instances of my_func in the RTL for hardware kernel to 1

```
void top { a, b, c, d} {
#pragma HLS ALLOCATION instances=my_func limit=1 function
 ...
 my_func(a,b); //my_func_1
 my_func(a,c); //my_func_2
 my_func(a,d); //my_func_3
 ...
}
```

Example2: Limits the number of multiplier operation used in the implementation of the function *my_func* to 1

- Limit does NOT apply outside the function
- Alternatively, ***inline*** the sub-function can also do similar job

```
void my_func(data_t angle) {
#pragma HLS allocation instances=mul limit=1 operation
...
}
```

Example

#pragma HLS allocation instances=<list> limit=<value> <type>

```
#include "example.h"

void example (
  unsigned int in[N],
  short a,
  short b,
  unsigned int c,
  unsigned int out[N]
  ) {

   unsigned int x, y;
   unsigned int tmp1, tmp2, tmp3;

for_Loop: for (unsigned int i=0 ; i < N; i++) {
#pragma HLS allocation instances=func limit=1 function
        x = in[i];
        tmp1 = func(1, 2);
        tmp2 = func(2, 3);
        tmp3 = func(1, 4);

        y = a*x + b + squared(c) + tmp1 + tmp2 + tmp3;

        out[i] = y;
      }
}

unsigned int squared(unsigned int a)
{
  unsigned int res = 0;
  res = a*a;
  return res;
}
```

Pragma HLS allocation

```
Timing:
  * Summary:
  +--------+----------+----------+------------+
  |  Clock |  Target  | Estimated| Uncertainty|
  +--------+----------+----------+------------+
  |ap_clk  | 25.00 ns | 7.401 ns |   3.12 ns  |
  +--------+----------+----------+------------+

Latency:
  * Summary:
  +---------+---------+----------+----------+-----+-----+---------+
  |  Latency (cycles) |  Latency (absolute) |  Interval | Pipeline|
  |   min   |   max   |    min   |    max   | min | max |   Type  |
  +---------+---------+----------+----------+-----+-----+---------+
  |      121|      121| 3.025 us | 3.025 us |  121|  121|   none  |
  +---------+---------+----------+----------+-----+-----+---------+
```

```
================================================================
== Utilization Estimates
================================================================
* Summary:
+---------------------+---------+-------+---------+---------+-----+
|         Name        | BRAM_18K| DSP48E|    FF   |   LUT   | URAM|
+---------------------+---------+-------+---------+---------+-----+
|DSP                  |        -|      -|        -|        -|    -|
|Expression           |        -|      5|        0|      169|    -|
|FIFO                 |        -|      -|        -|        -|    -|
|Instance             |        -|      -|        -|        -|    -|
|Memory               |        -|      -|        -|        -|    -|
|Multiplexer          |        -|      -|        -|       30|    -|
|Register             |        -|      -|       85|        -|    -|
+---------------------+---------+-------+---------+---------+-----+
|Total                |        0|      5|       85|      199|    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 Latency

#pragma HLS latency min=<int> max=<int>

- Specifies a minimum or maximum latency value, or both, for the completion of functions, loops, and regions
  - *min=<int>*: minimum latency for the function, loop, or region of code
  - *max=<int>*: maximum latency for the function, loop, or region of code

- **Latency:** # of CLK cycles required to produce an output

- **Function latency:** # of CLK cycles required for the function to compute all output values and return

- **Loop latency**: # of CLK cycles to execute all iterations of the loop

Pragma HLS Latency

#pragma HLS latency min=<int> max=<int>

- HLS always tries to minimize latency in the design

- When LATENCY pragma is specified
  - ***Min < Latency < Max***: Constraint is satisfied, No further optimization
  - ***Latency < min***: It extends latency to the specified value, potentially increasing sharing
  - ***Latency > max***: Increases effort to achieve the constraints
    - Still unsuccessful: issue a warning & produce design with the smallest achievable latency in excess of maximum

Pragma HLS Latency - Example

#pragma HLS latency min=<int> max=<int>

Example-1: Function foo is specified to have a minimum latency of 4 and a maximum latency of 8

```
int foo(char x, char a, char b, char c) {
  #pragma HLS latency min=4 max=8
  char y;
  y = x*a+b+c;
  return y
}
```

Example-2: loop_1 is specified to have a maximum latency of 12

```
void foo (num_samples, ...) {
  int i;
  ...
  loop_1: for(i=0;i< num_samples;i++) {
  #pragma HLS latency max=12
    ...
    result = a + b;
  }
}
```

Example-3: Creates a code region and groups signals that need to change in the same clock cycle by specifying zero latency

```
// create a region { } with a latency = 0
{
  #pragma HLS LATENCY max=0 min=0
  *data = 0xFF;
  *data_vld = 1;
}
```

Pragma HLS Latency - Example

```
void example (
  unsigned int in[N],
  short a,
  short b,
  unsigned int c,
  unsigned int out[N]
  ) {

   unsigned int x, y;
   unsigned int tmp1, tmp2, tmp3;

for_Loop: for (unsigned int i=0 ; i < N; i++) {
#pragma HLS latency min=4
        x = in[i];
        tmp1 = func(1, 2);
        tmp2 = func(2, 3);
        tmp3 = func(1, 4);

        y = a*x + b + squared(c) + tmp1 + tmp2 + tmp3;

        out[i] = y;
      }
}
```

Pragma HLS Latency - Results

```
Timing:
  * Summary:
  +--------+----------+----------+------------+
  |  Clock |  Target  | Estimated| Uncertainty|
  +--------+----------+----------+------------+
  |ap_clk  | 25.00 ns | 7.401 ns |   3.12 ns  |
  +--------+----------+----------+------------+

Latency:
  * Summary:
  +---------+---------+----------+----------+-----+-----+---------+
  |  Latency (cycles) |  Latency (absolute) |  Interval | Pipeline|
  |   min   |   max   |    min   |    max   | min | max |   Type  |
  +---------+---------+----------+----------+-----+-----+---------+
  |      301|      301| 7.525 us | 7.525 us |  301|  301|   none  |
  +---------+---------+----------+----------+-----+-----+---------+

  + Detail:
      * Instance:
      N/A

      * Loop:
      +-----------+---------+---------+----------+-----------+-----------+------+----------+
      |           |  Latency (cycles) | Iteration|  Initiation Interval  | Trip |          |
      |  Loop Name|   min   |   max   |  Latency |  achieved |   target  | Count| Pipelined|
      +-----------+---------+---------+----------+-----------+-----------+------+----------+
      |- for_Loop |      300|      300|         5|          -|          -|    60|    no    |
      +-----------+---------+---------+----------+-----------+-----------+------+----------+
```

```
================================================================
= Utilization Estimates
================================================================
 Summary:
+---------------------+---------+-------+---------+---------+-----+
|         Name        | BRAM_18K| DSP48E|    FF   |   LUT   | URAM|
+---------------------+---------+-------+---------+---------+-----+
|DSP                  |        -|      -|        -|        -|    -|
|Expression           |        -|      5|        0|      154|    -|
|FIFO                 |        -|      -|        -|        -|    -|
|Instance             |        -|      -|        -|        -|    -|
|Memory               |        -|      -|        -|        -|    -|
|Multiplexer          |        -|      -|        -|       47|    -|
|Register             |        -|      -|      120|        -|    -|
+---------------------+---------+-------+---------+---------+-----+
|Total                |        0|      5|      120|      201|    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|
+---------------------+---------+-------+---------+---------+-----+
```

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

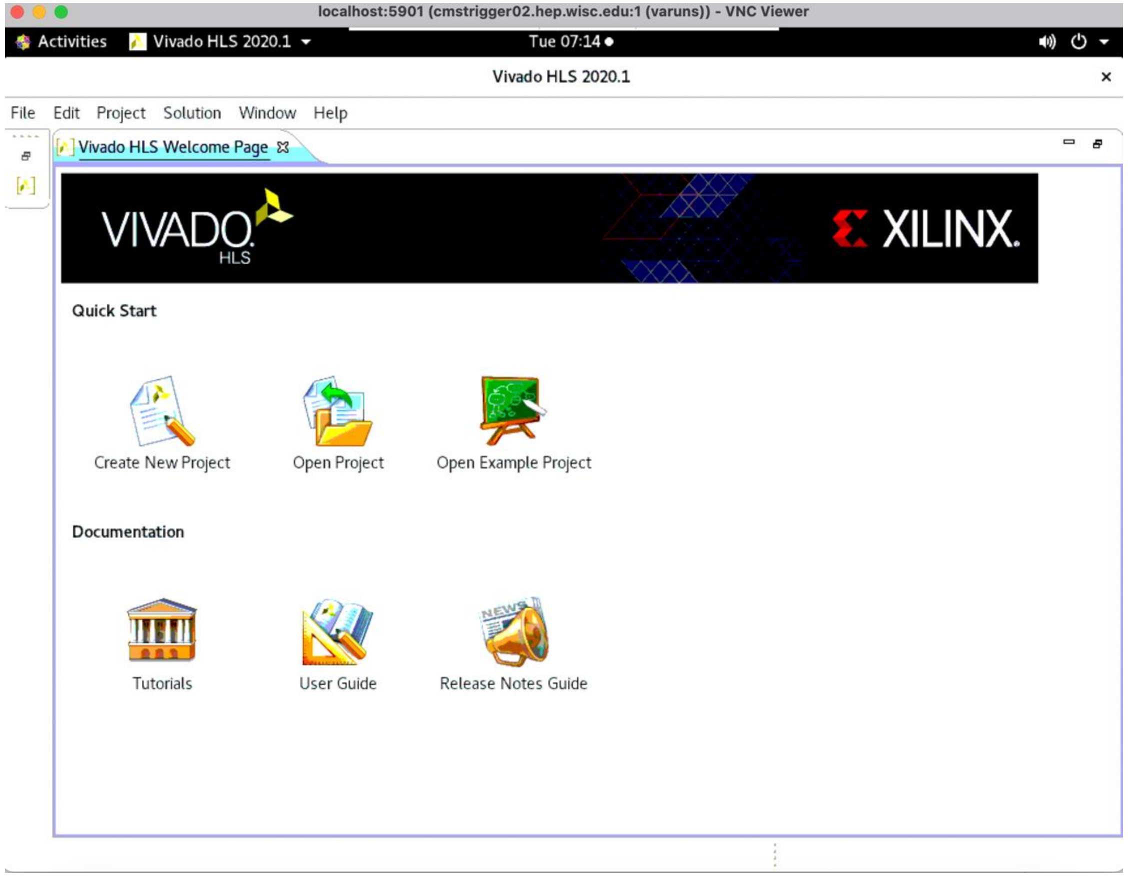
Reminder: HLS Setup

- 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