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

FPGA module training

More on FPGAs

Lecture-6: March 10th 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

Today

- Unsupported C/C++ constructs
- Data types
- HLS Pragmas
  - Interface

TAC-HEP 2026

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

```
#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
```

```
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 2026

Data Types

Data Types

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

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

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

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

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

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

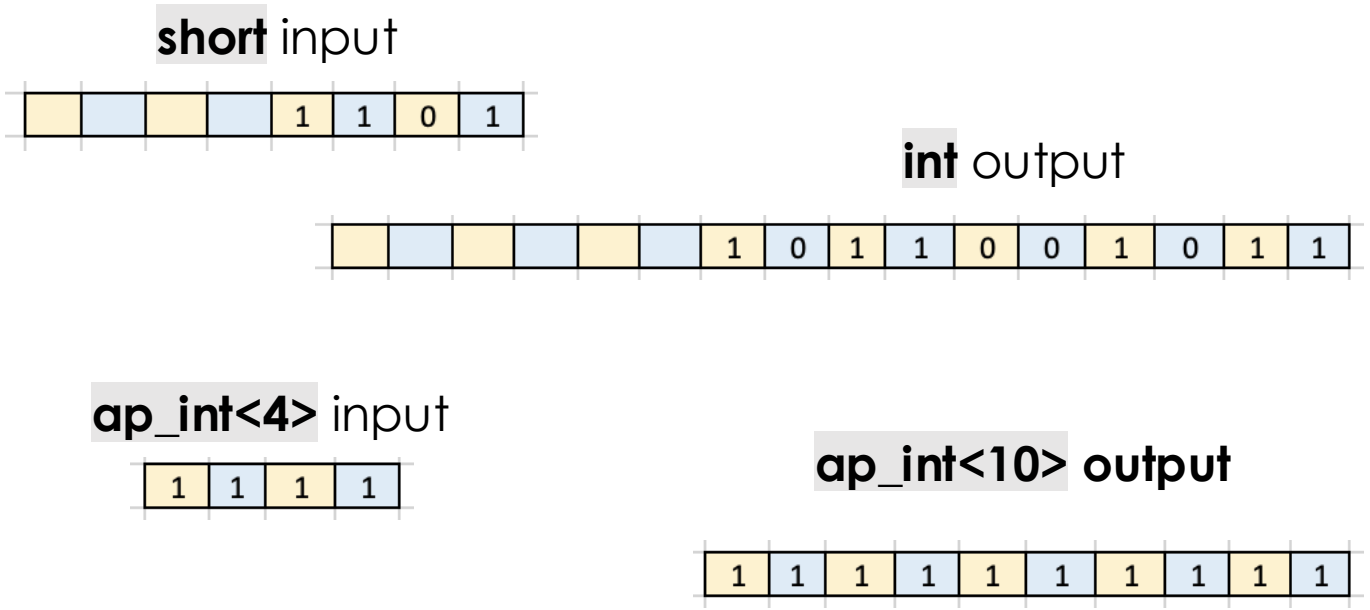
  `float`, `double`

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

Arbitrary precision

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

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



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

Arbitrary precision

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

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

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

Simple arithmetic example

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

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

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

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

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

Data-Type (w/o Arbitrary precision)

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

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


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

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

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

Data-Type (w/o Arbitrary precision)

+ Detail:
  * Instance:

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

  * DSP48E:

Instance	Module	Expression
basic_arith_mul_mdEe_U3	basic_arith_mul_mdEe	i0 * i1

  * Expression:

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

  * Multiplexer:

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

  * Register:

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

== Interface

* Summary:

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

Precise data types

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

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

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

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

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

Data-Type (w/ Arbitrary precision)

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

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


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

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

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

Data-Type (w/ Arbitrary precision)

```
+ Detail:
    * Instance:
    +-----------------------+---------------------+---------+-------+-----+-----+-----+
    |        Instance       |        Module       | BRAM_18K| DSP48E|  FF | LUT | URAM|
    +-----------------------+---------------------+---------+-------+-----+-----+-----+
    |basic_arith_ap_sdcud_U2|basic_arith_ap_sdcud |        0|      0|  274|  148|    0|
    |basic_arith_ap_srbkb_U1|basic_arith_ap_srbkb |        0|      0|  406|  247|    0|
    +-----------------------+---------------------+---------+-------+-----+-----+-----+
    |Total                  |                     |        0|      0|  680|  395|    0|
    +-----------------------+---------------------+---------+-------+-----+-----+-----+

    * DSP48E:
    +-----------------------+---------------------+-----------+
    |        Instance       |        Module       | Expression|
    +-----------------------+---------------------+-----------+
    |basic_arith_ap_mudEe_U3|basic_arith_ap_mudEe |  i0 * i1  |
    +-----------------------+---------------------+-----------+
```

```
    * Expression:
    +-------------+----------+-------+---+----+------------+------------+
    | Variable Name| Operation| DSP48E| FF| LUT| Bitwidth P0| Bitwidth P1|
    +-------------+----------+-------+---+----+------------+------------+
    |out2_V       |     +    |      0|  0|  20|          13|          13|
    +-------------+----------+-------+---+----+------------+------------+
    |Total        |          |      0|  0|  20|          13|          13|
    +-------------+----------+-------+---+----+------------+------------+

    * Multiplexer:
    +-----------+-----+-----------+-----+-----------+
    |    Name   | LUT | Input Size| Bits| Total Bits|
    +-----------+-----+-----------+-----+-----------+
    |ap_NS_fsm  |  169|         38|    1|         38|
    +-----------+-----+-----------+-----+-----------+
    |Total      |  169|         38|    1|         38|
    +-----------+-----+-----------+-----+-----------+

    * Register:
    +-----------+----+----+-----+-----------+
    |    Name   | FF | LUT| Bits| Const Bits|
    +-----------+----+----+-----+-----------+
    |ap_CS_fsm  |  37|   0|   37|          0|
    +-----------+----+----+-----+-----------+
    |Total      |  37|   0|   37|          0|
    +-----------+----+----+-----+-----------+
```

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

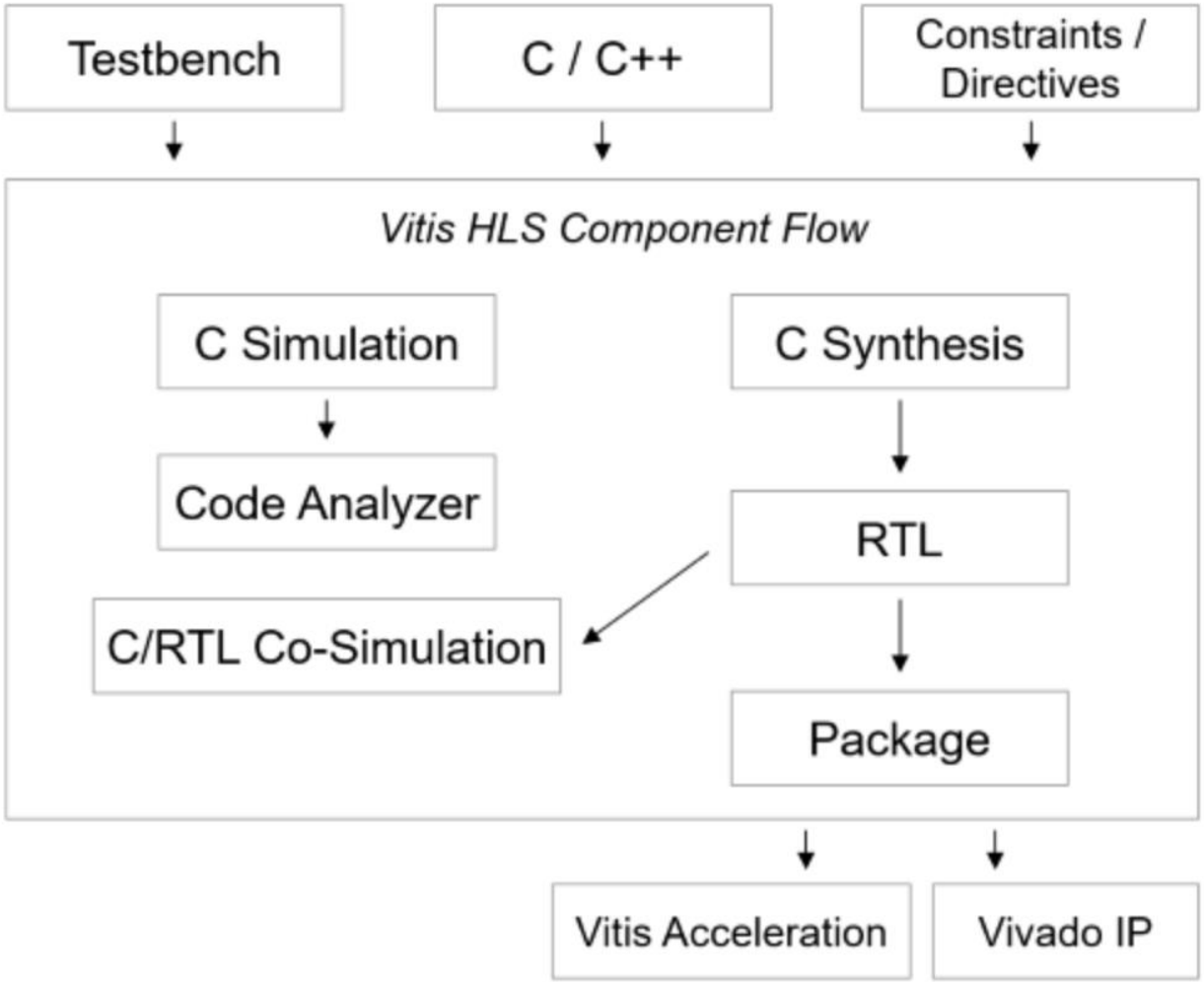
TAC-HEP 2026

HLS Pragmas

Steps: C++ function ➜ HLS Component

1. Architect the algorithm based on the Design Principles.
2. (C-Simulation) Verify the functionality of the C/C++ code with the C/C++ test bench.
3. (Code Analyzer) Analyze the performance, parallelism, and legality of the C/C++ code.
4. (C-Synthesis) Generate the RTL using the v++ compiler.
5. (C/RTL Co-Simulation) Verify the RTL code generated using the C/C++ test bench.
6. (Package) Review the HLS synthesis reports and implementation timing reports.
7. Re-run previous steps until performance goals are met.

HLS Component Development Flow



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

Pragma HLS interface

C-argument type	Paradigm	Interface protocol (I/O/Inout)
Scalar(pass by value)	Register	AXI4-Lite (`s_axilite`)
Array	Memory	AXI4 Memory Mapped (`m_axi`)
Pointer to array	Memory	`m_axi`
Pointer to scalar	Register	`s_axilite`
Reference	Register	`s_axilite`
hls::stream	Stream	AXI4-Stream (`axis`)

Homework # 2

- 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

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 (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