The Abdus Salam International Centre for Theoretical Physics

High-Level Synthesis: Bridging Software and Hardware

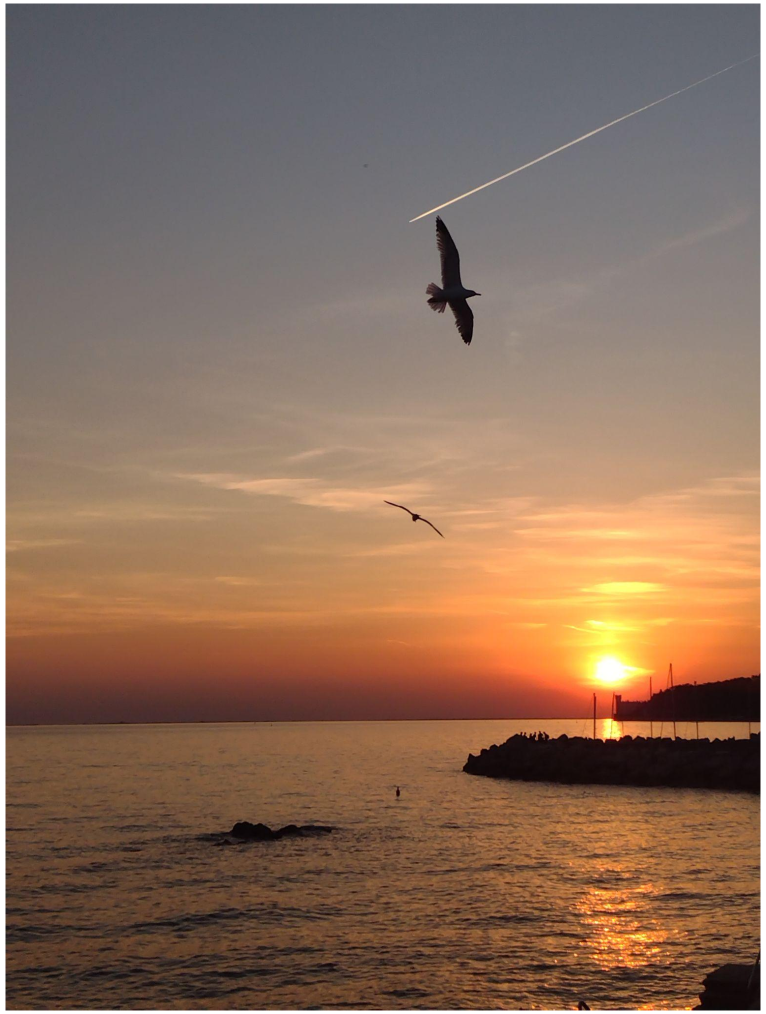
Romina Soledad Molina, Ph.D.
MLab-STI, ICTP

Perú - Online - 2025 -

Universidad Tecnológica del Perú

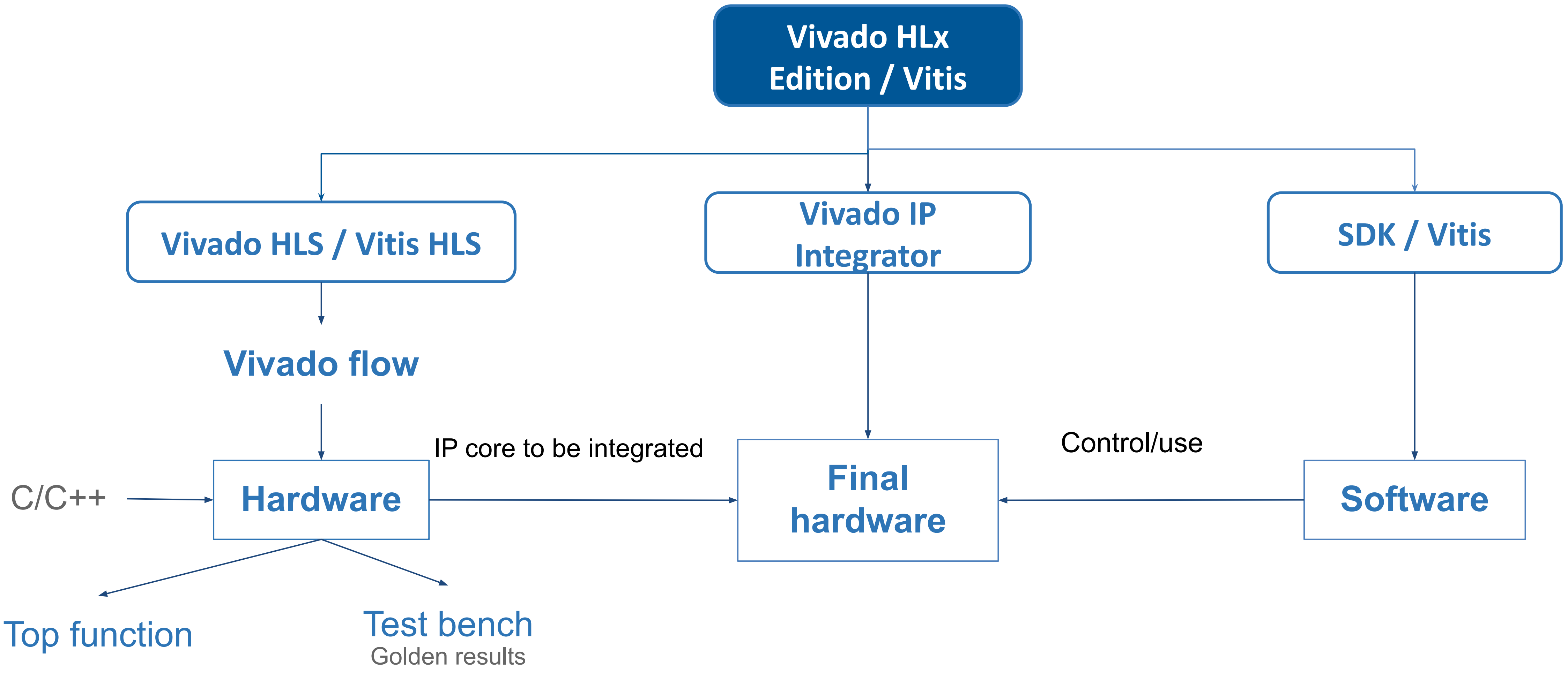
Outline

- Introduction.
- High-level synthesis.
- HLS Component Development Flow.
- C-to-RTL Conversion.
- Language Support.
- Hardware design: Directives/Optimizations.
- Vitis HLS GUI.
- Demo: HLS and Matrix Multiplication.

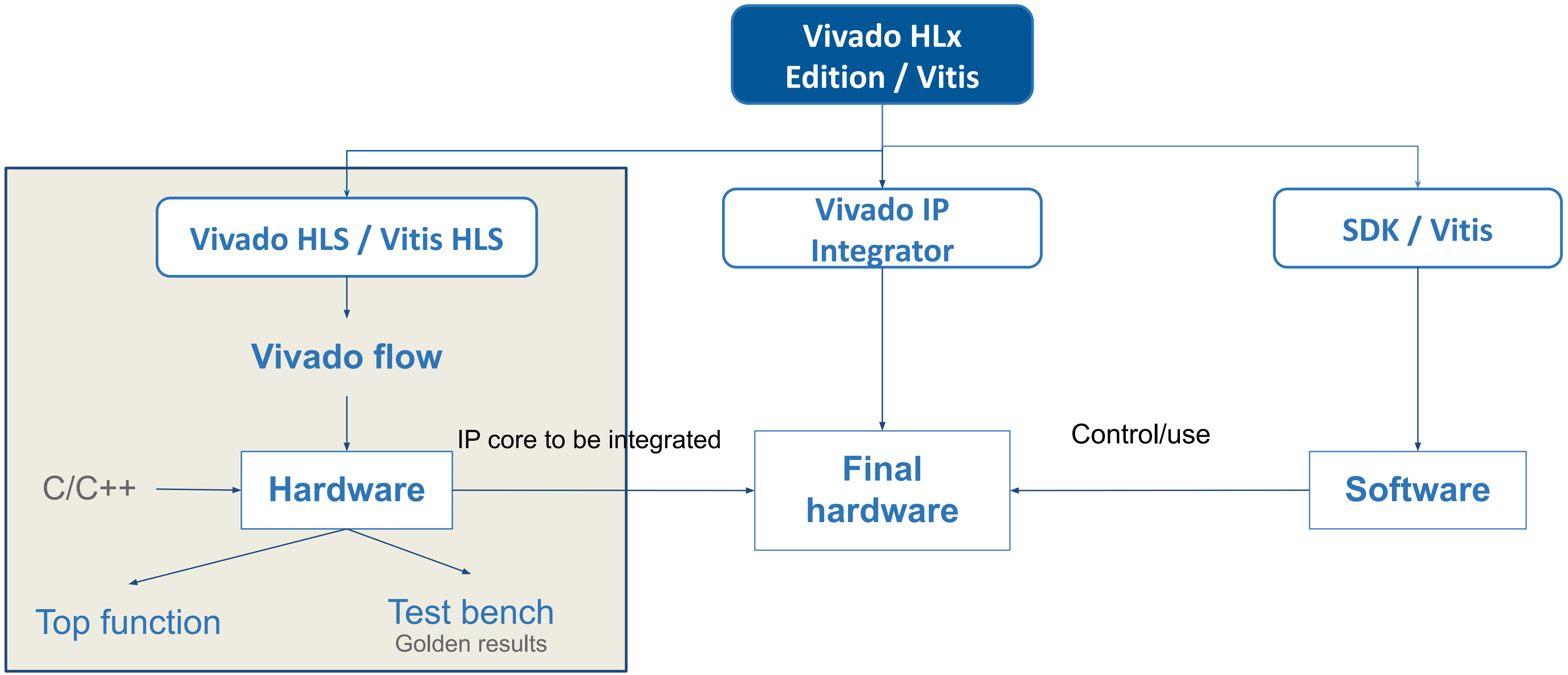


Introduction

Introduction

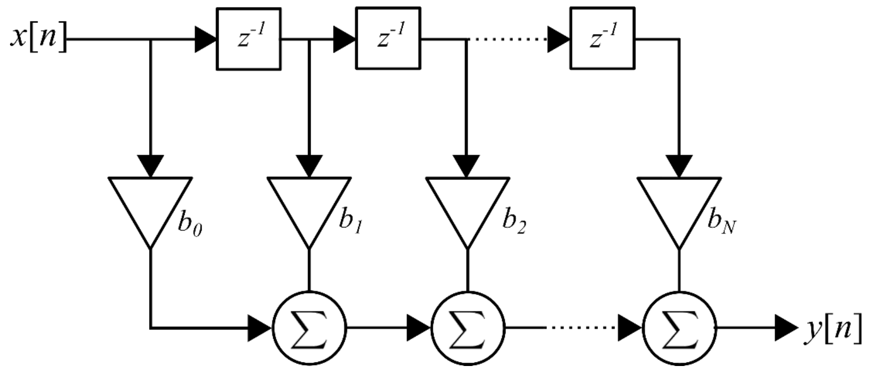


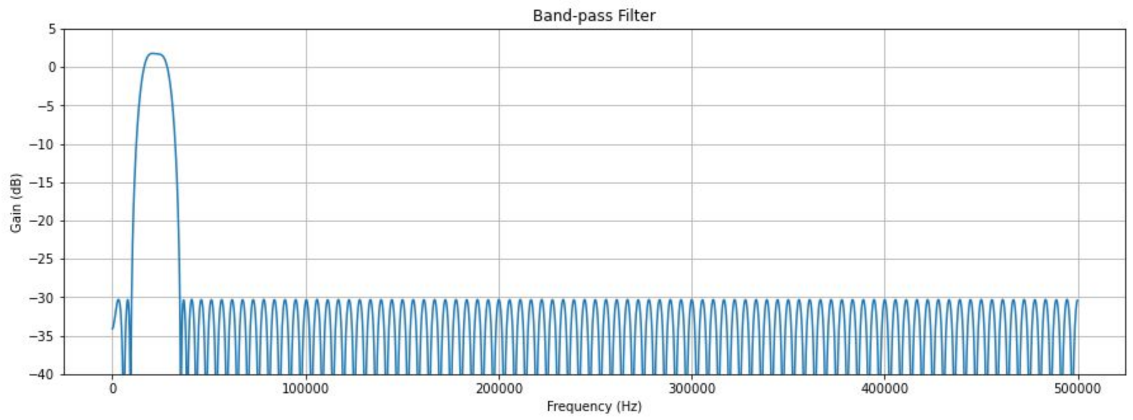
Introduction



Introduction

- **Traditional RTL FIR Filter Design**
  - Define interfaces
  - Define architecture
    - FSM
    - Datapath
  - Write RTL code
  - Write RTL test bench.
- **The design choice is already made.**





Introduction

- *Question: Why the need of High-Level Synthesis tools?*

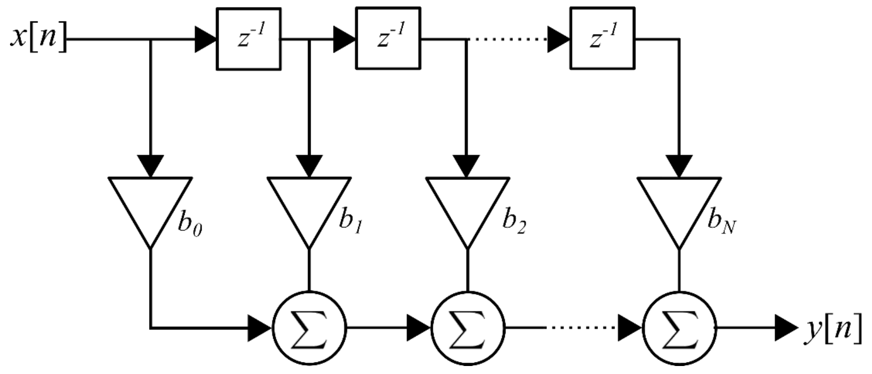
Introduction

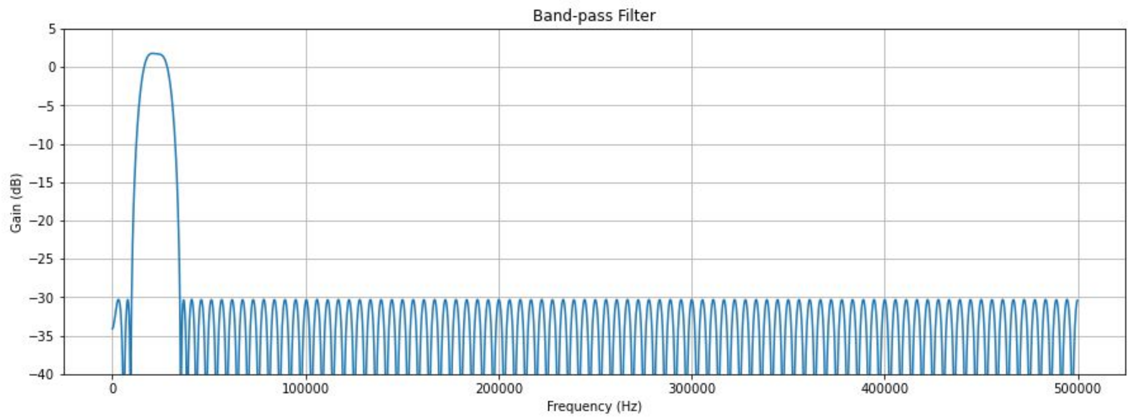
- *Question: Why the need of High-Level Synthesis tools?*
    - Some reasons could be:
        - Productivity boosting.
        - Trend to use FPGA as hardware accelerators.
        - Reduce Time-To-Market.
        - Design Space Exploration.
        - Early metric estimations.
        - Functionality verification through C-based test bench.
        - Reuse.

Introduction

- **HLS-based FIR Filter Design**

```
void fir (data_t *y,  data_t *x ) {
        static data_t shift_reg[N];
        acc_t acc;
        data_t data;
        int i;
        acc=0;
        Shift_Accum_Loop: for (i=N-1;i>=0;i--) {
                if (i==0) {
                        shift_reg[0]= *x;
                        data = *x;
                } else {
                        shift_reg[i]=shift_reg[i-1];
                        data = shift_reg[i];
                 }
        acc+=data*firCoeff[i];
        }
        acc = acc >> 16;
        *y=acc;
}
```





High-Level Synthesis

Introduction

- **High-Level Synthesis**
    - It provides the facility to create RTL from a high level of abstraction.

Introduction

- **High-Level Synthesis**
  - It provides the facility to create RTL from a high level of abstraction.
  - Several implementations are possible from the same source code description.

Introduction

- **High-Level Synthesis**
  - It provides the facility to create RTL from a high level of abstraction.
  - Several implementations are possible from the same source code description.
  - Implements the design based on defaults and user applied directives.

Introduction

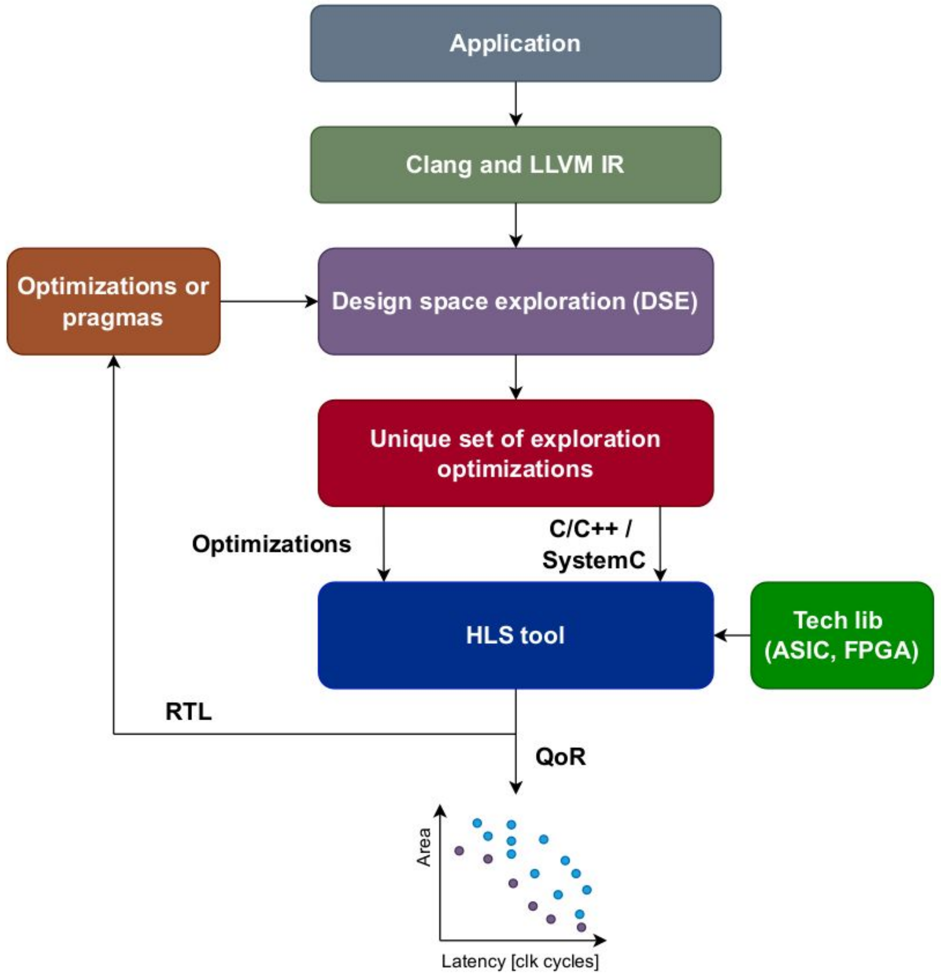
- **High-Level Synthesis**
    - It provides the facility to create RTL from a high level of abstraction.
    - Several implementations are possible from the same source code description.
    - Implements the design based on defaults and user applied directives.
    - It allows the optimization of the input code using directives to:
        - Reduce latency
        - Improve performance and throughput
        - Reduce resource utilization

Introduction

- **High-Level Synthesis**
  - It provides the facility to create RTL from a high level of abstraction.
  - Several implementations are possible from the same source code description.
  - Implements the design based on defaults and user applied directives.
  - It allows the optimization of the input code using directives to:
    - Reduce latency.
    - Improve performance and throughput.
    - Reduce resource utilization.

Without optimization, HLS tool will look to minimize latency and improve concurrency.

Design space exploration



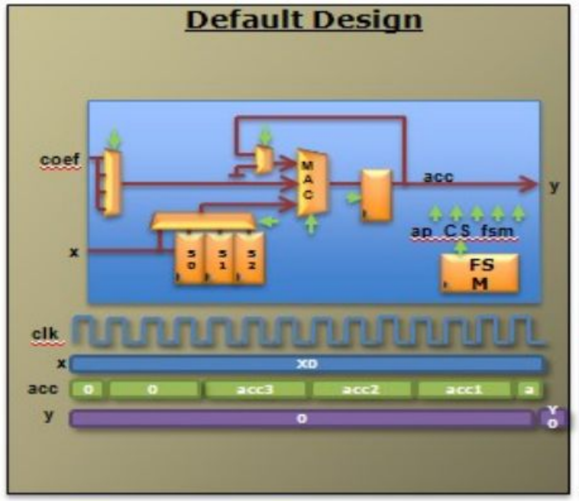
Design space exploration

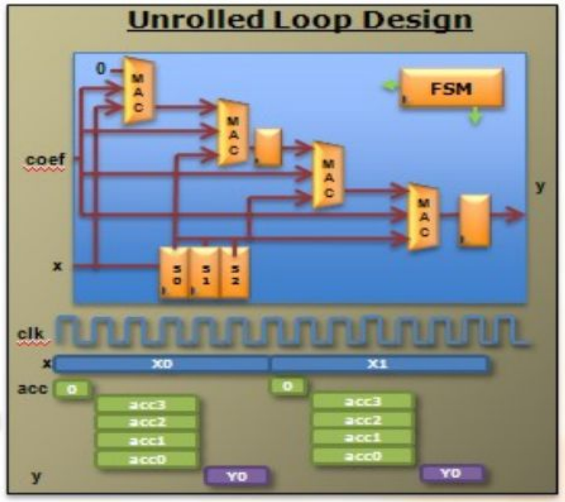
```
…
  loop: for (i=3;i>=0;i--) {
     if (i==0) {
       acc+=x*c[0];
       shift_reg[0]=x;
     } else {
        shift_reg[i]=shift_reg[i-1];
        acc+=shift_reg[i]*c[i];
     }
```

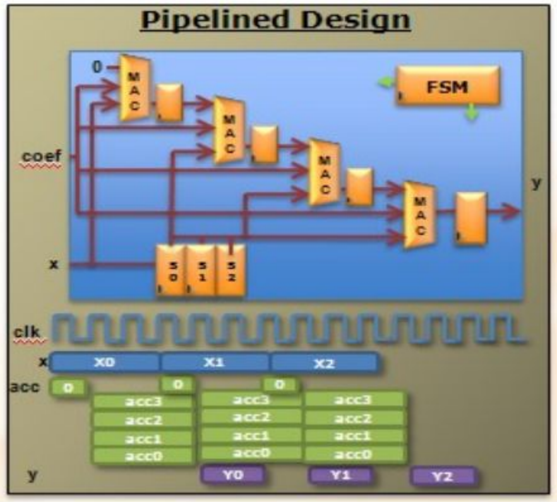
Same hardware is used for each loop iteration :
- Small area
- Long latency
- Low throughput

Different hardware for each loop iteration :
- Higher area
- Short latency
- Better throughput

Different iterations executed concurrently:
- Higher area
- Short latency
- Best throughput

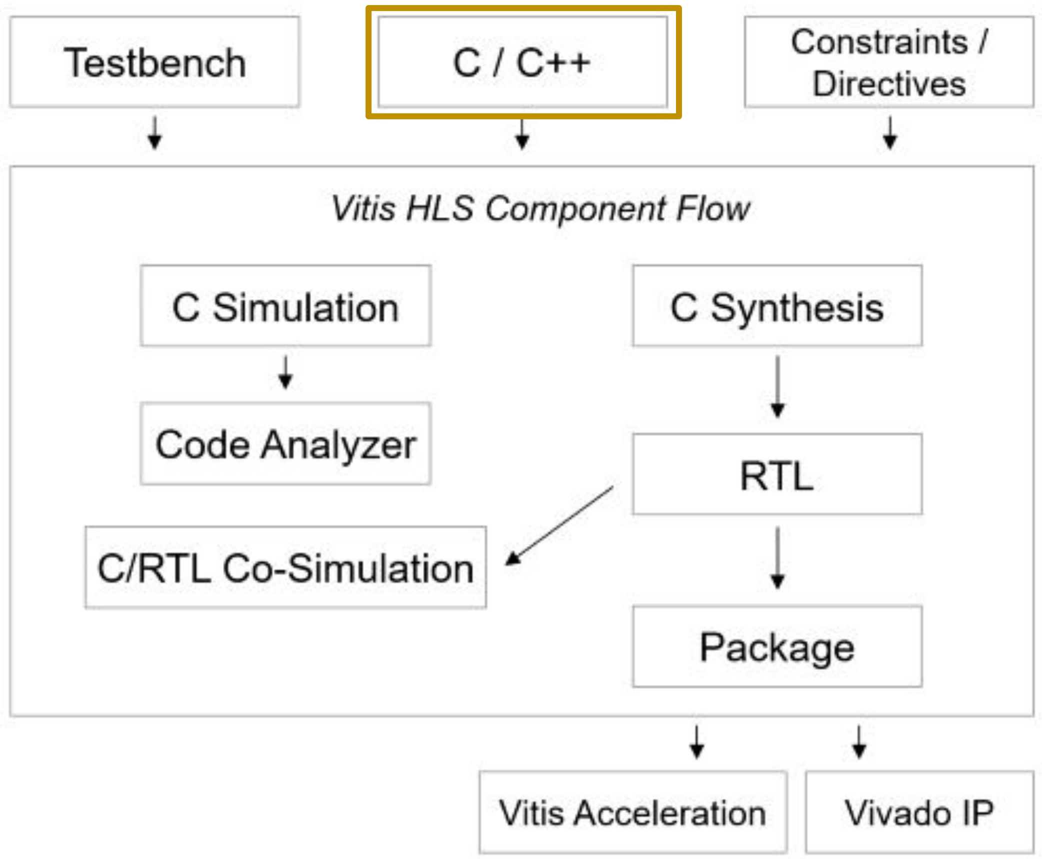






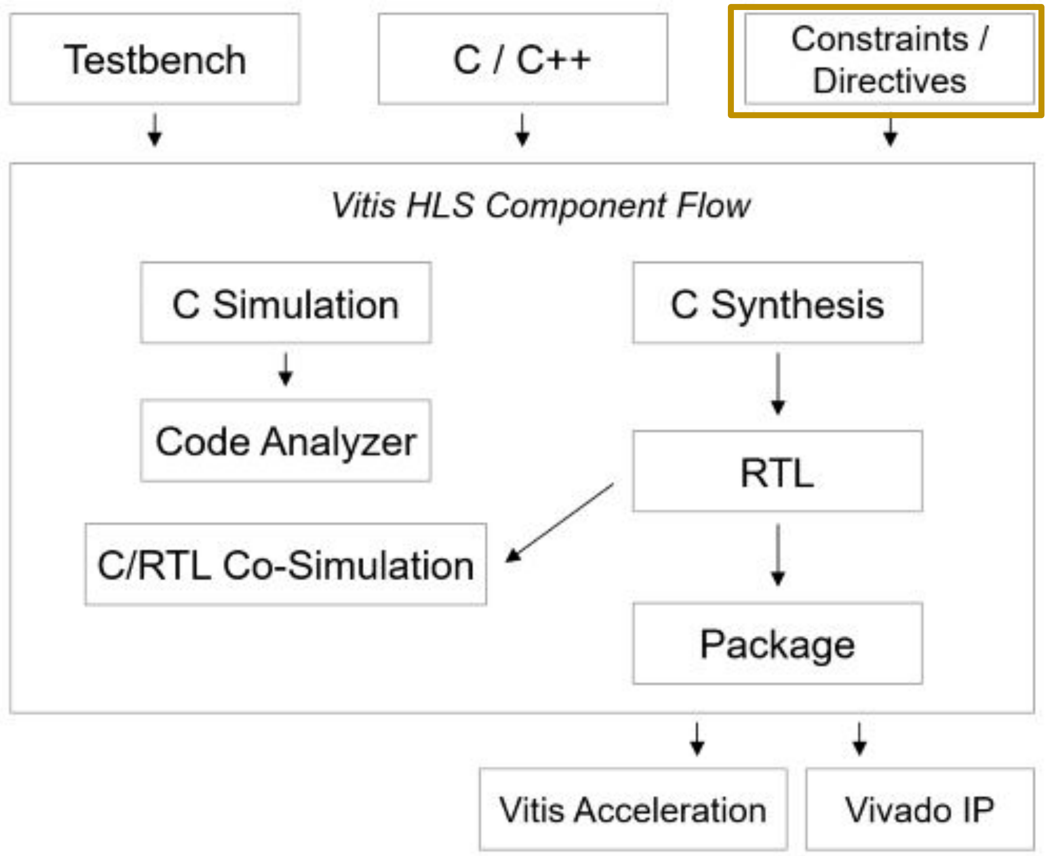
HLS Component Development Flow

HLS Component Development Flow



https://docs.amd.com/r/en-US/ug1399-vitis-hls/Introduction-to-Vitis-HLS-Components

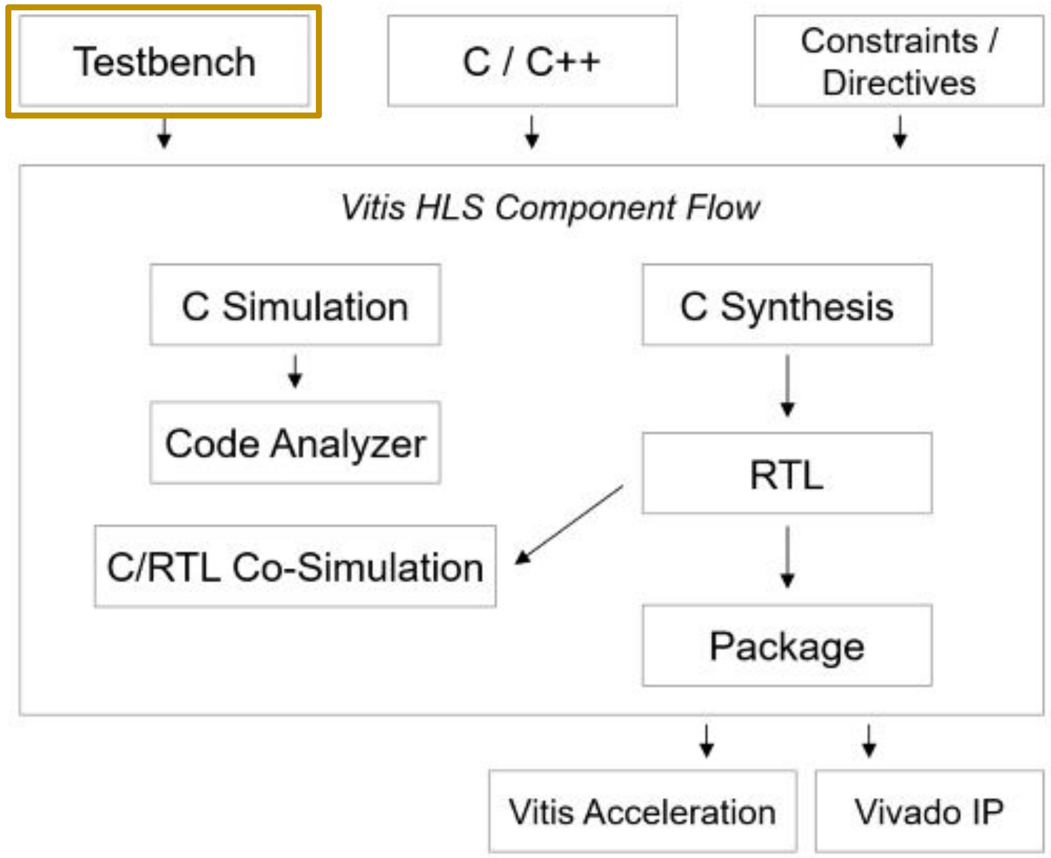
HLS Component Development Flow



https://docs.amd.com/r/en-US/ug1399-vitis-hls/Introduction-to-Vitis-HLS-Components

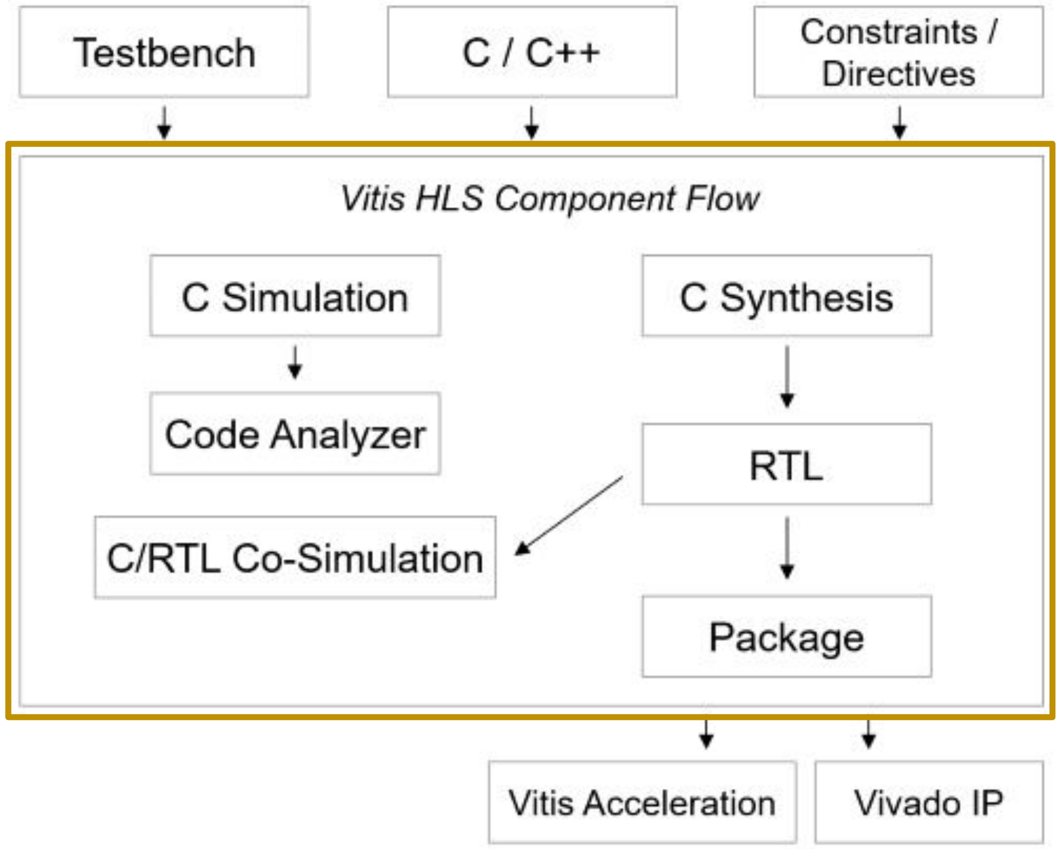
ICTP The Abdus Salam International Centre for Theoretical Physics

HLS Component Development Flow



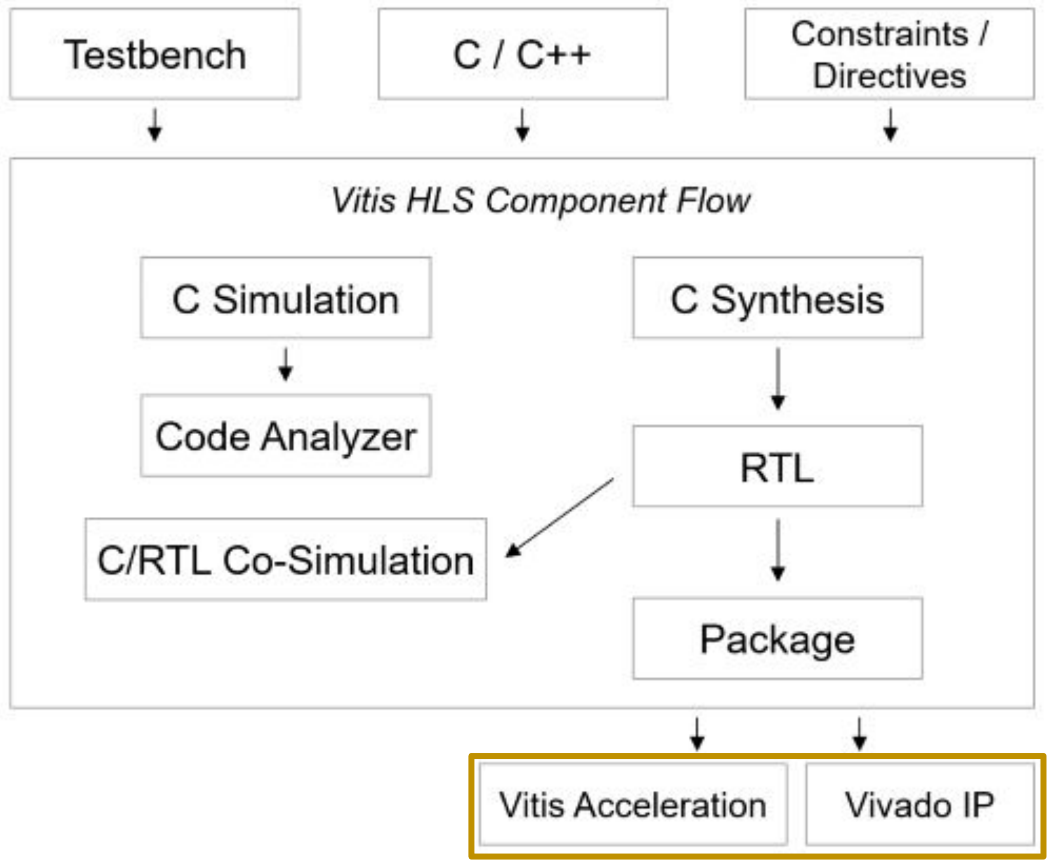
https://docs.amd.com/r/en-US/ug1399-vitis-hls/Introduction-to-Vitis-HLS-Components

HLS Component Development Flow



https://docs.amd.com/r/en-US/ug1399-vitis-hls/Introduction-to-Vitis-HLS-Components

HLS Component Development Flow



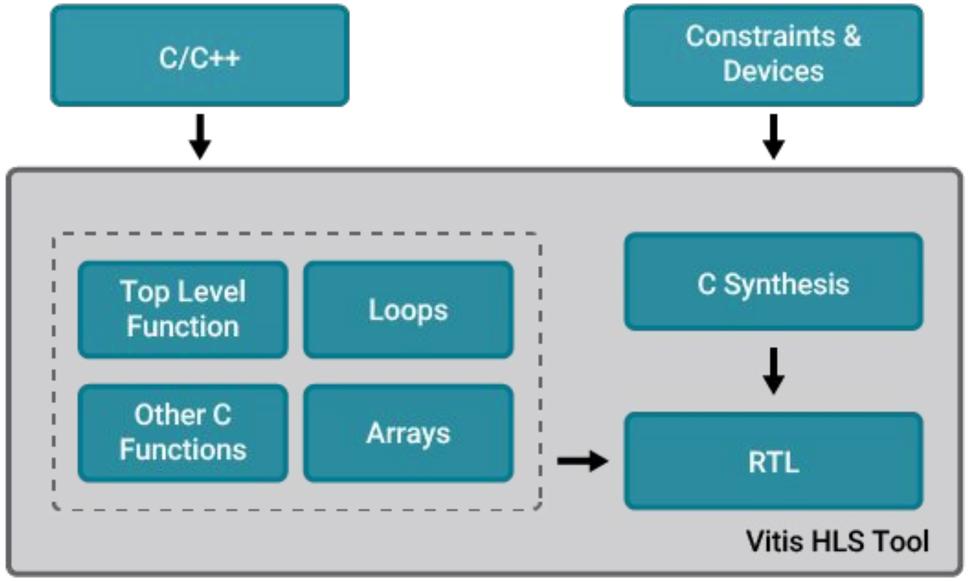
https://docs.amd.com/r/en-US/ug1399-vitis-hls/Introduction-to-Vitis-HLS-Components

C-to-RTL Conversion

C-to-RTL Conversion

The Vitis HLS tool synthesizes different parts of C code differently:

- **Top-level function arguments of the C/C++ code** are synthesized into **RTL I/O** ports and are automatically implemented with an interface synthesis hardware protocol -> **Only one top function.**



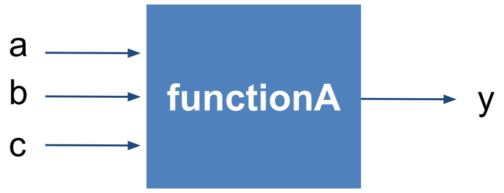
Source: https://www.amd.com/en/products/software/adaptive-socs-and-fpgas/vitis/vitis-hls.html

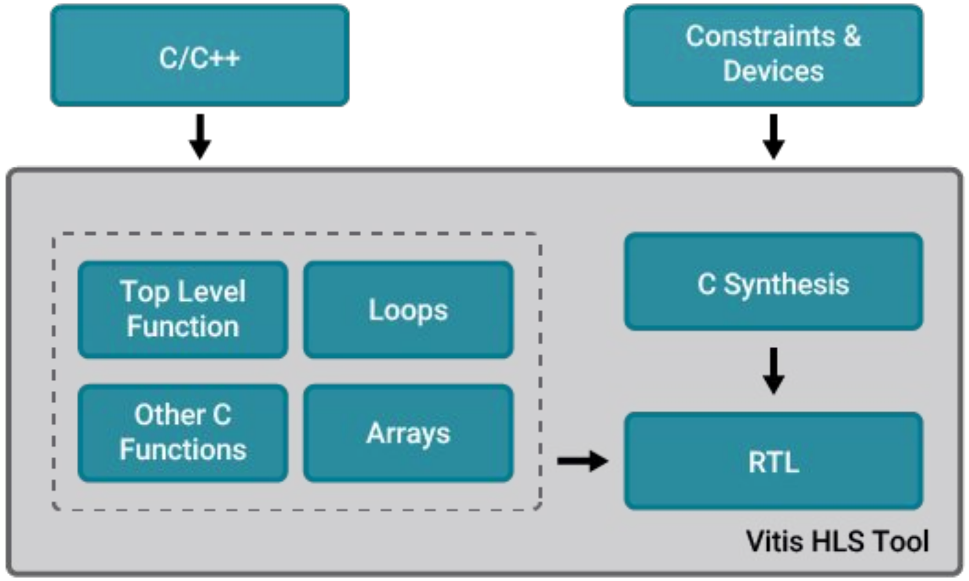
C-to-RTL Conversion

The Vitis HLS tool synthesizes different parts of C code differently:

- **Top-level function arguments of the C/C++ code** are synthesized into **RTL I/O** ports and are automatically implemented with an interface synthesis hardware protocol → **Only one top function.**

```
void functionA(char x, char a, char b, char c, char y) {
        char tmp = b*2;
        tmp = x*a;
         y = tmp+c;
}
```



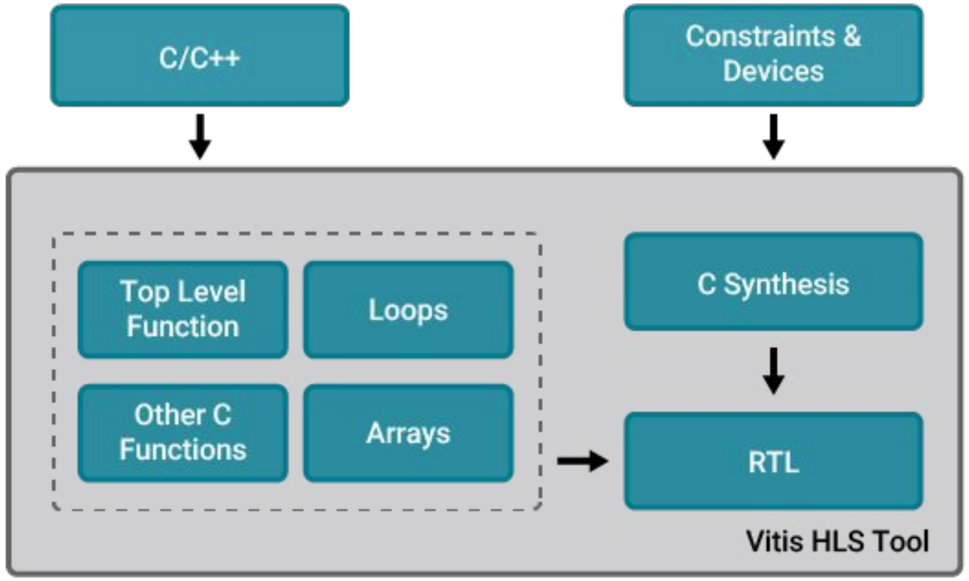


Source: https://www.amd.com/en/products/software/adaptive-socs-and-fpgas/vitis/vitis-hls.html

C-to-RTL Conversion

The Vitis HLS tool synthesizes different parts of C code differently:

- **Top-level function arguments of the C/C++ code** are synthesized into **RTL I/O** ports and are automatically implemented with an interface synthesis hardware protocol → **Only one top function.**
- **Other C functions are synthesized to RTL blocks**—maintaining the design hierarchy.
- **Arrays in the C code** can be targeted to any memory resource, such as BRAM, LUTRAM, and URAM.

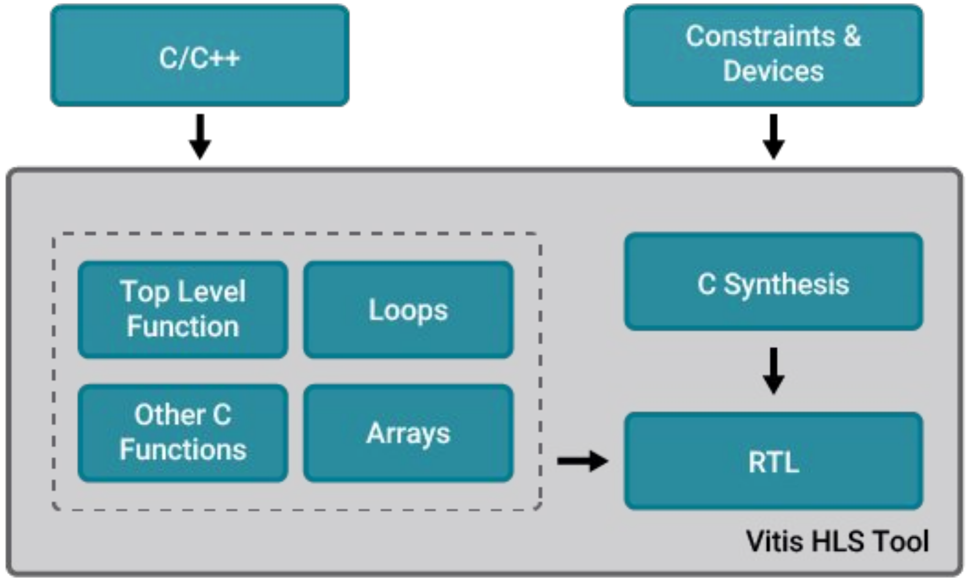


Source: https://www.amd.com/en/products/software/adaptive-socs-and-fpgas/vitis/vitis-hls.html

C-to-RTL Conversion

- **Performance metrics**, such as latency, initiation interval, loop iteration latency, and resource utilization, can be reviewed with synthesis reports.

- Vitis HLS tool **pragmas and optimization directives** allow for configuring the synthesis results for the C/C++ code.



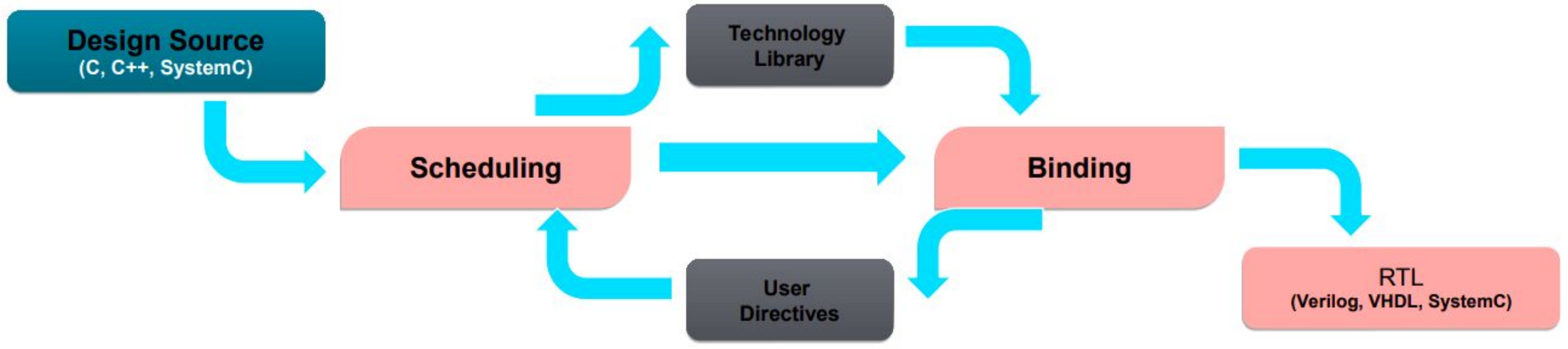
Source: https://www.amd.com/en/products/software/adaptive-socs-and-fpgas/vitis/vitis-hls.html

The Abdus Salam International Centre for Theoretical Physics

C-to-RTL Conversion

Scheduling and binding

- **Scheduling** determines in which clock cycle an operation will occur.
- **Binding** determines which library cell is used for each operation.
  - For example, for a functional unit like adder, there can be many options like ripple-carry adder, carry-look-ahead-adder etc.



Source: https://www.amd.com/en/products/software/adaptive-socs-and-fpgas/vitis/vitis-hls.html

C-to-RTL Conversion

Scheduling and binding

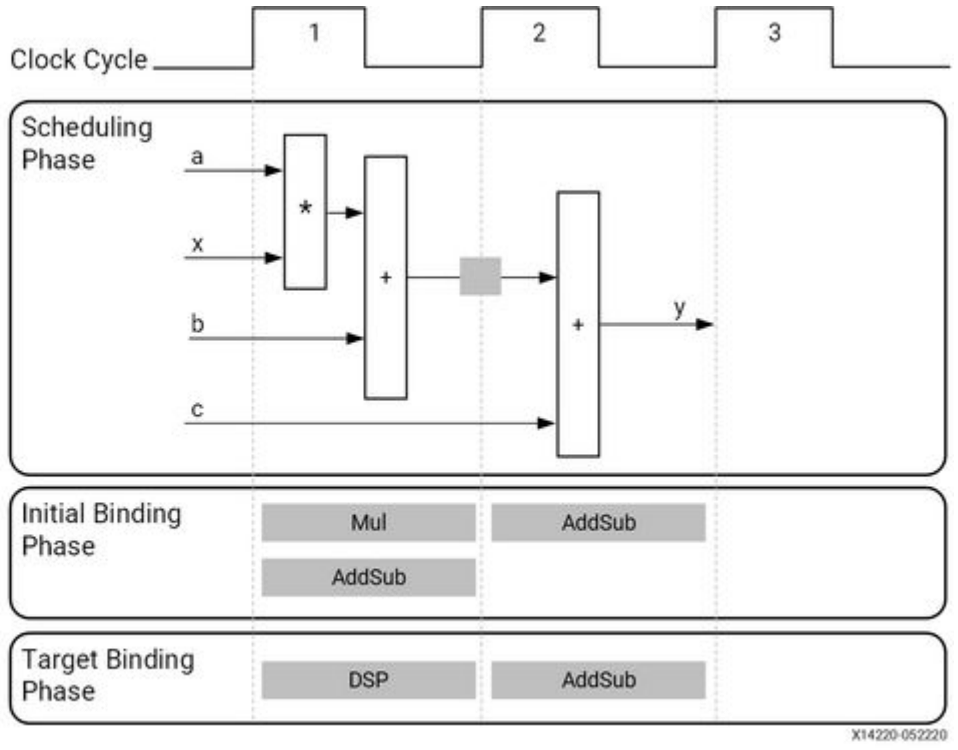
```
int foo(char x, char a, char b, char c) {
    char y;
    y = x*a+b+c;
    return y;
}
```

Source: https://docs.amd.com/r/2020.2-English/ug1399-vitis-hls/Scheduling-and-Binding-Example

C-to-RTL Conversion

Scheduling and binding

```
int foo(char x, char a, char b, char c) {
        char y;
        y = x*a+b+c;
        return y;
}
```



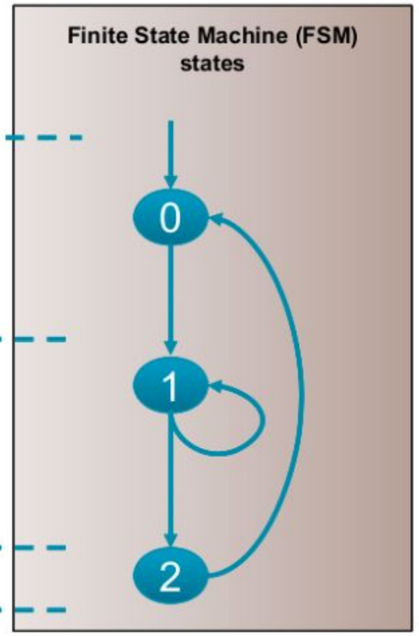
Source: https://docs.amd.com/r/2020.2-English/ug1399-vitis-hls/Scheduling-and-Binding-Example

C-to-RTL Conversion

Code

```
void fir (
  data_t *y,
  coef_t c[4],
  data_t x
  ) {

  static data_t shift_reg[4];
  acc_t acc;
  int i;

  acc=0;
  loop: for (i=3;i>=0;i--) {
    if (i==0) {
      acc+=x*c[0];
      shift_reg[0]=x;
    } else {
      shift_reg[i]=shift_reg[i-1];
      acc+=shift_reg[i]*c[i];
    }
  }
  *y=acc;
}
```

Control Behavior



Function Start

For-Loop Start

For-Loop End

Function End

From any C code example ..

The loops in the C code correlated to states of behavior

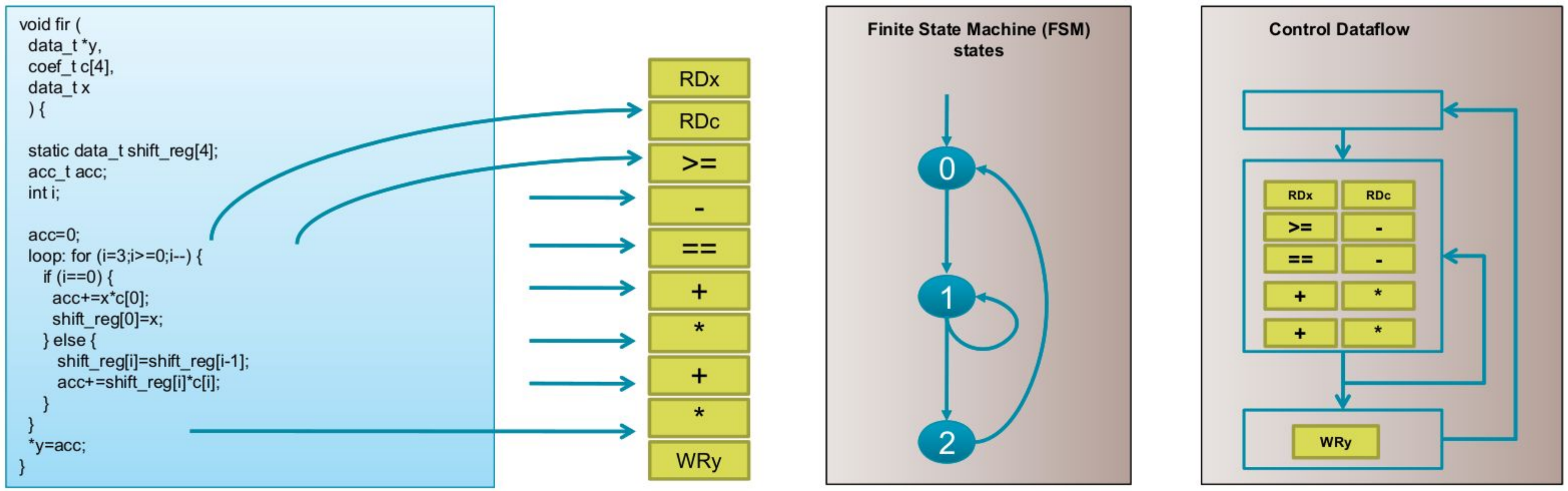
This behavior is extracted into a hardware state machine

© Copyright 2016 Xilinx

XILINX ALL PROGRAMMABLE.

C-to-RTL Conversion

Code | **Operations** | **Control Behavior** | **Control & Datapath Behavior**



```
void fir (
  data_t *y,
  coef_t c[4],
  data_t x
  ) {

  static data_t shift_reg[4];
  acc_t acc;
  int i;

  acc=0;
  loop: for (i=3;i>=0;i--) {
    if (i==0) {
      acc+=x*c[0];
      shift_reg[0]=x;
    } else {
      shift_reg[i]=shift_reg[i-1];
      acc+=shift_reg[i]*c[i];
    }
  }
  *y=acc;
}
```

From any C code example ..

Operations are extracted…

The control is known

A unified control dataflow behavior is created.

© Copyright 2016 Xilinx

XILINX ALL PROGRAMMABLE.

Language Support

Language Support

- Vivado/Vitis HLS supports C, C++, SystemC, and OpenCL API C kernel.
- Supports arbitrary precision types for all input languages.
- Floating point support.
- Support for OpenCV functions.

Source: https://www.amd.com/en/products/software/adaptive-socs-and-fpgas/vitis/vitis-hls.html

Language Support

- The function must contain the entire functionality of the design.
- None of the functionality can be performed by system calls to the operating system.
- The C/C++ constructs must be of a fixed or bounded size.
- The implementation of those constructs must be unambiguous.

Source: https://docs.amd.com/r/2020.2-English/ug1399-vitis-hls/Vitis-HLS-Coding-Styles

The Abdus Salam International Centre for Theoretical Physics

Language Support

- The function must contain the entire functionality of the design.
- None of the functionality can be performed by system calls to the operating system.
- The C/C++ constructs must be of a fixed or bounded size.
- The implementation of those constructs must be unambiguous.

- Unsupported C/C++ Constructs:
  - System calls.
  - Dynamic memory usage.
  - Recursive functions.

Source: https://docs.amd.com/r/2020.2-English/ug1399-vitis-hls/Vitis-HLS-Coding-Styles

Data type precision

- **Standard C Types**
  - Integers:
    - long long (64 bits)
    - int (32 bits)
    - Short (16 bits)
  - Characters:
    - char (8 bits)
  - Floating Point:
    - Float (32 bits)
    - Double (64 bits)
- **Arbitrary Precision Types**
  - C:
    - ap_(u)int
  - C++:
    - ap_(u)int
    - ap_fixed
  - C++ / SystemC:
    - sc_(u)int
    - sc_fixed

Hardware design: Directives/Optimizations

Hardware design - Directives/Optimizations

Minimize latency: UNROLL, LOOP_FLATTEN, LOOP_MERGE.
Minimize throughput: DATAFLOW, PIPELINE.
Improve bottleneck: RESOURCE, ARRAY_PARTITION, ARRAY_RESHAPE.

Hardware design - Directives/Optimizations

Minimize latency: UNROLL, LOOP_FLATTEN, LOOP_MERGE.
Minimize throughput: DATAFLOW, PIPELINE.
Improve bottleneck: RESOURCE, ARRAY_PARTITION, ARRAY_RESHAPE.

```
#pragma HLS UNROLL
#pragma HLS PIPELINE
#pragma HLS ARRAY_PARTITION variable=layer3_out complete dim=0
```

Hardware design - Directives/Optimizations

Minimize latency: UNROLL, LOOP_FLATTEN, LOOP_MERGE.
Minimize throughput: DATAFLOW, PIPELINE.
Improve bottleneck: RESOURCE, ARRAY_PARTITION, ARRAY_RESHAPE.

```
#pragma HLS UNROLL
#pragma HLS PIPELINE
#pragma HLS ARRAY_PARTITION variable=layer3_out complete dim=0
```

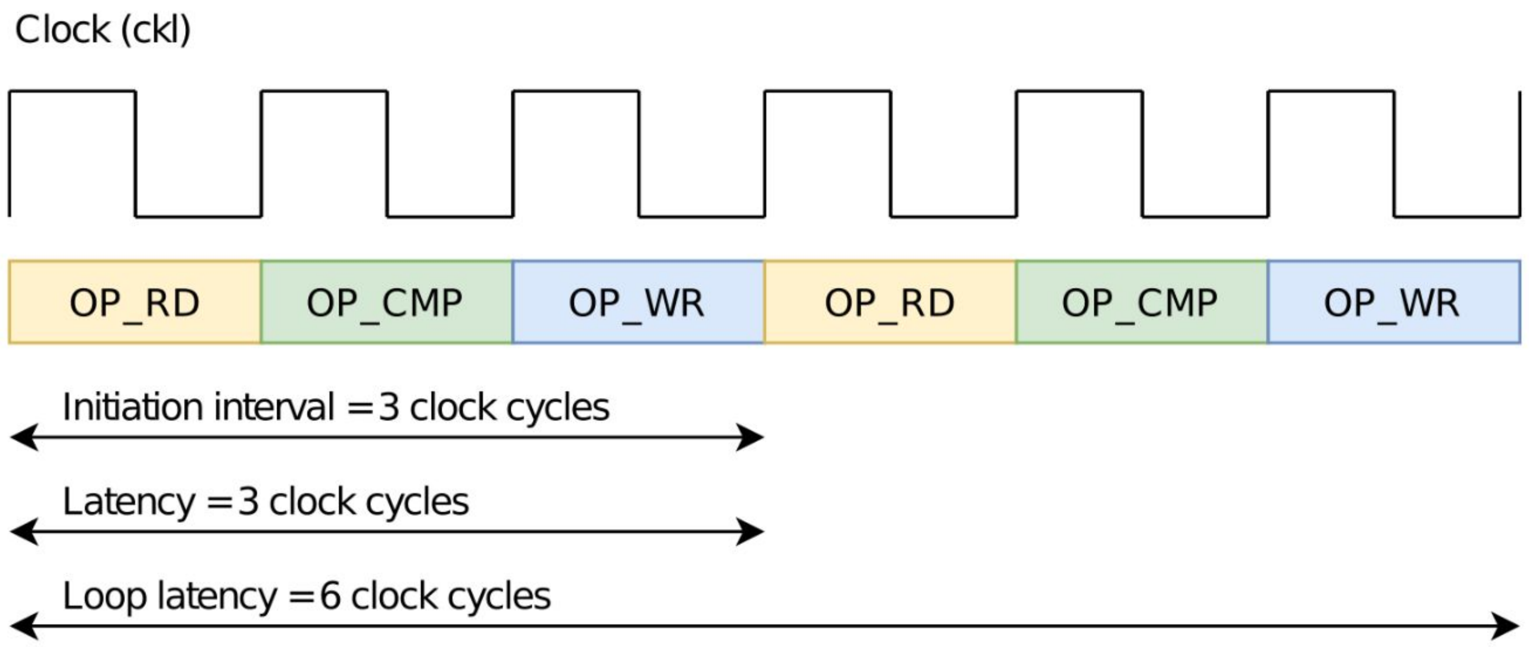
Source code: directives are included in the code.
Directive file: directives are specified in a separated file.

Hardware design - Directives/Optimizations

Loop handle

```
Loop_1: for(i=1; i<3; i++){
      OP_RD;
      OP_CMP;
      OP_WR;
}
```

OP_RD	Read
OP_CMP	Computation
OP_WR	Write



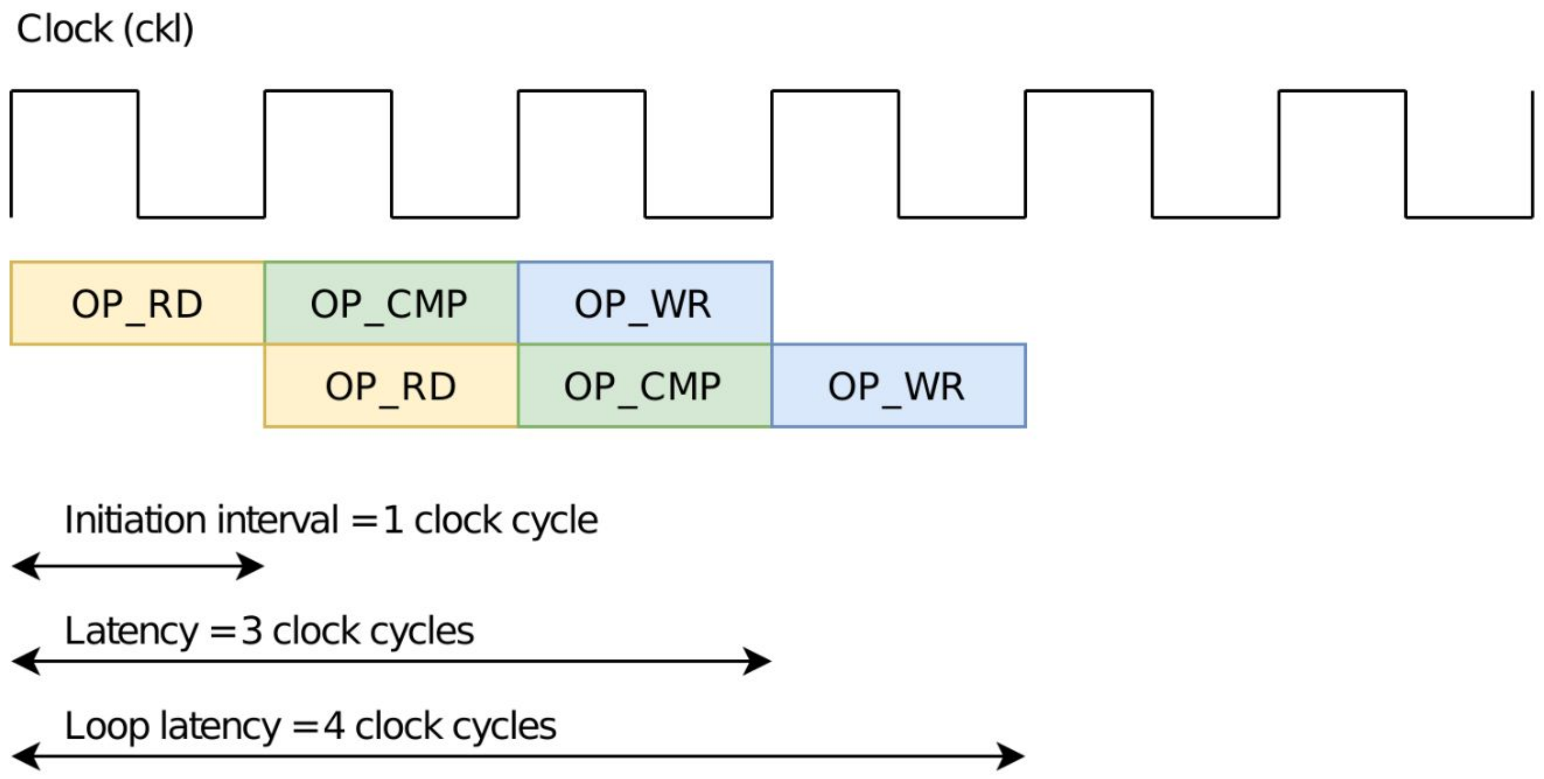
Hardware design - Directives/Optimizations

Loop handle

Loop + Pipeline

```
Loop_1: for(i=1; i<3; i++){
        OP_RD;
        OP_CMP;
        OP_WR;
}
```

🟨 OP_RD	Read
🟩 OP_CMP	Computation
🟦 OP_WR	Write



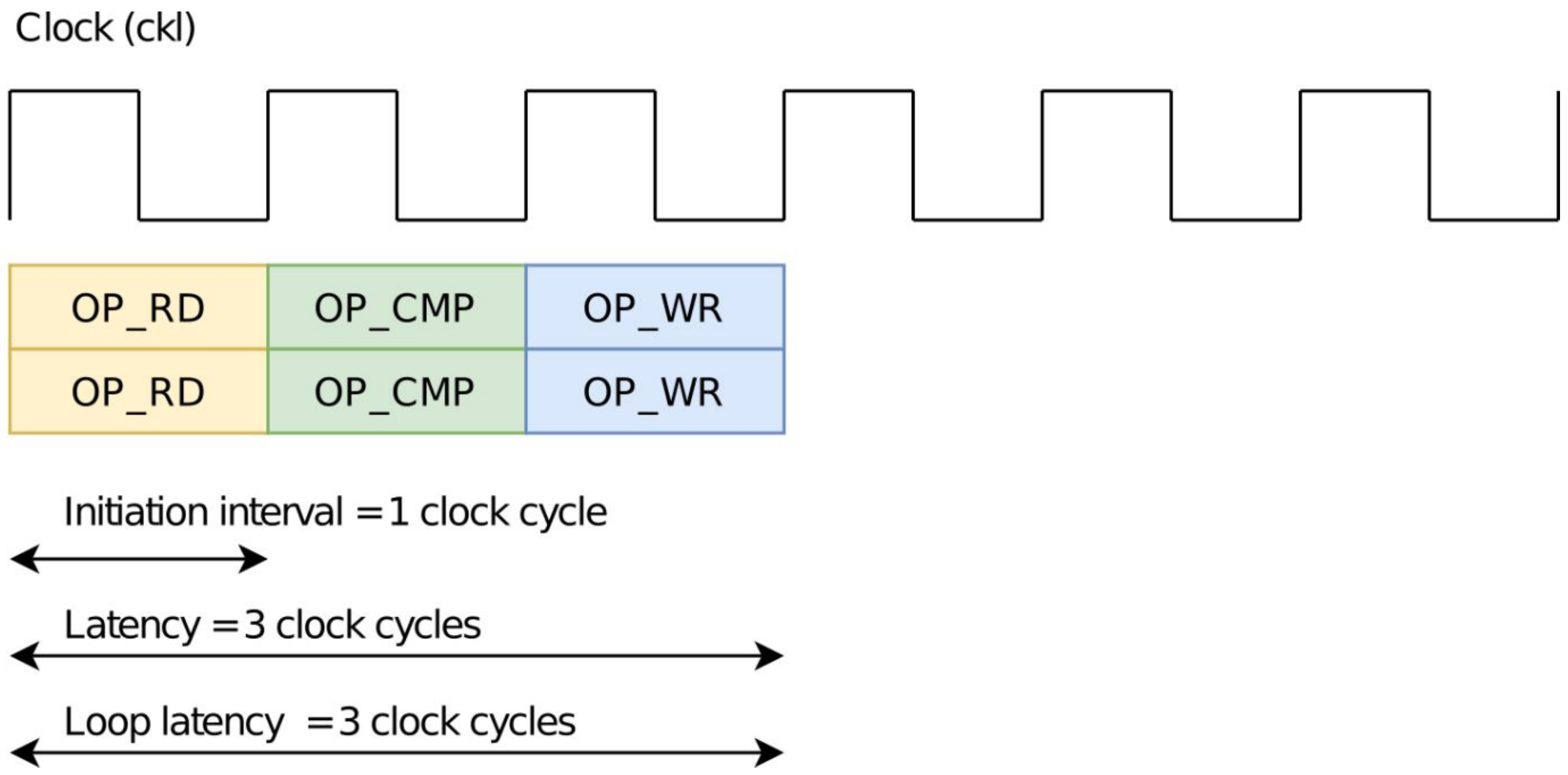
Hardware design - Directives/Optimizations

Loop handle

Loop + Unroll

```
Loop_1: for(i=1; i<3; i++){
      OP_RD;
      OP_CMP;
      OP_WR;
}
```

OP_RD	Read
OP_CMP	Computation
OP_WR	Write



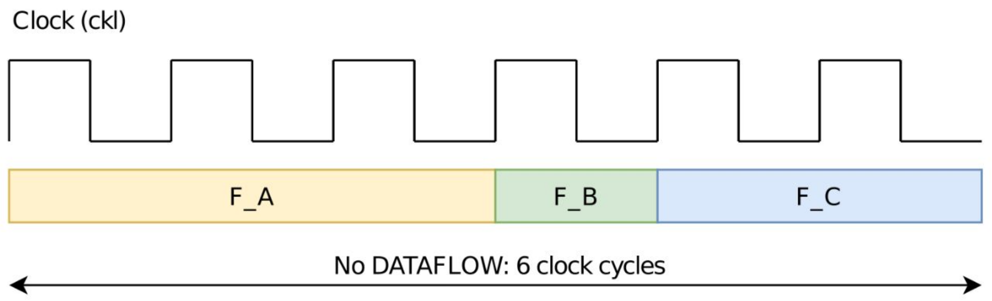
Hardware design - Directives/Optimizations

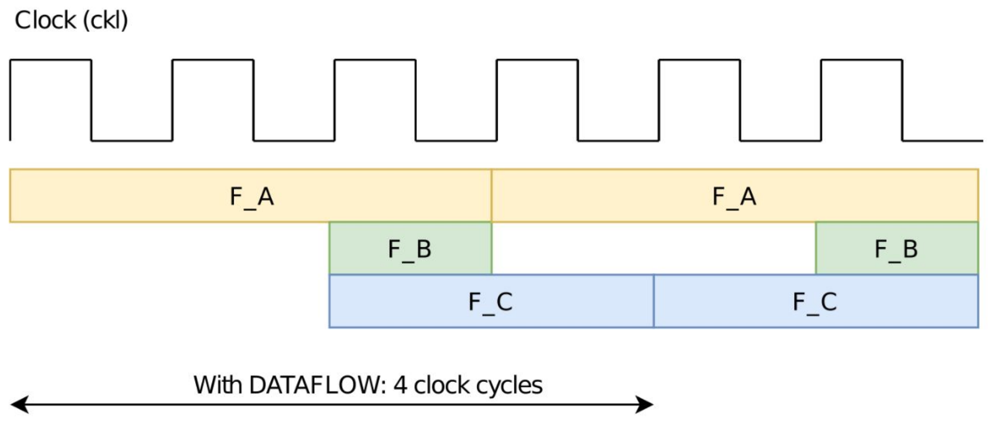
Loop handle

Loop + Dataflow

```
void function(a, b, c, d) {
    f_A(a, b, i1);
    f_B(c, i1, i2);
    f_C(i2, d);

return d;
}
```

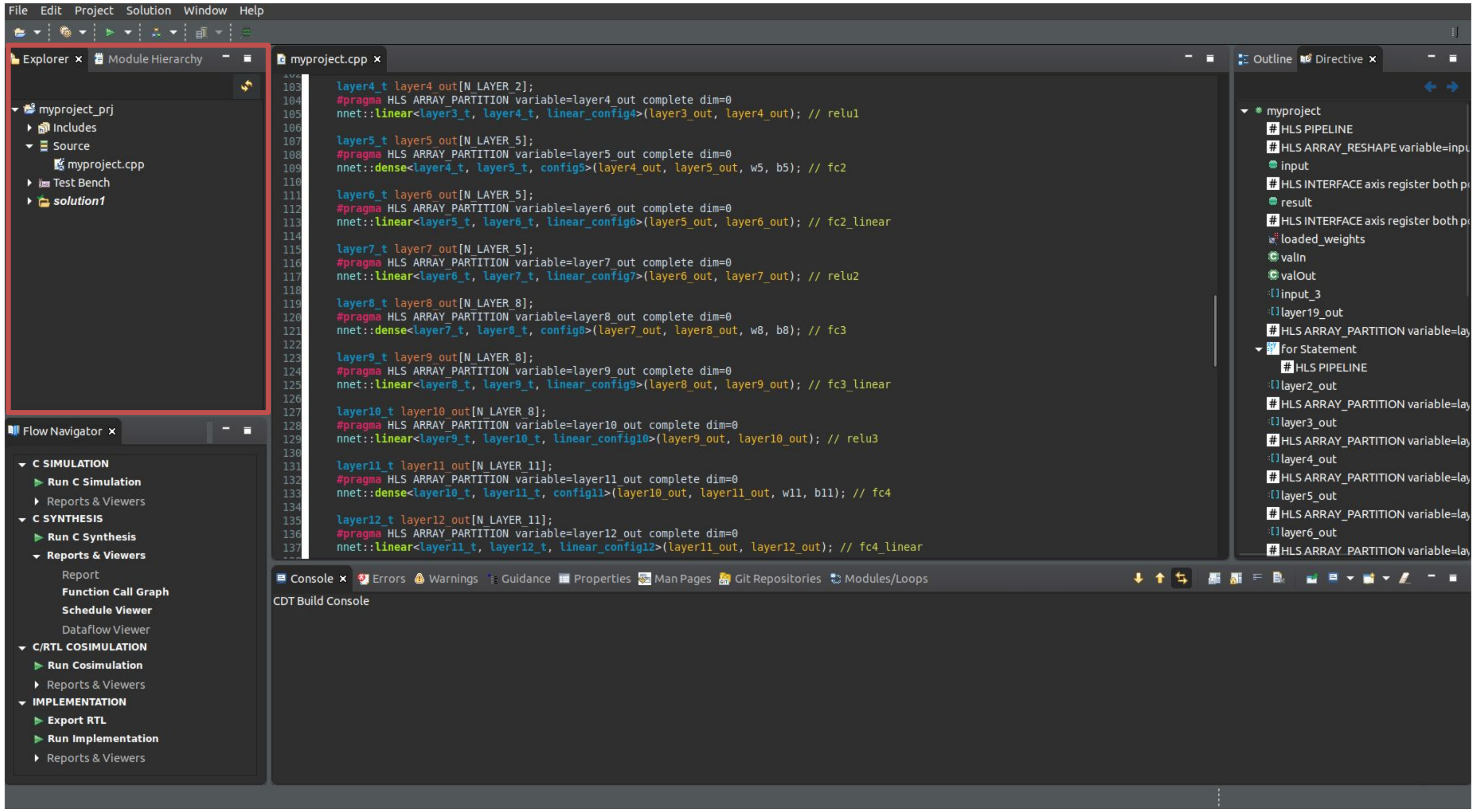
F_A	Function f_A(a, b, i1)
F_B	Function f_B(c, i1, i2)
F_C	Function f_C(i2, d)





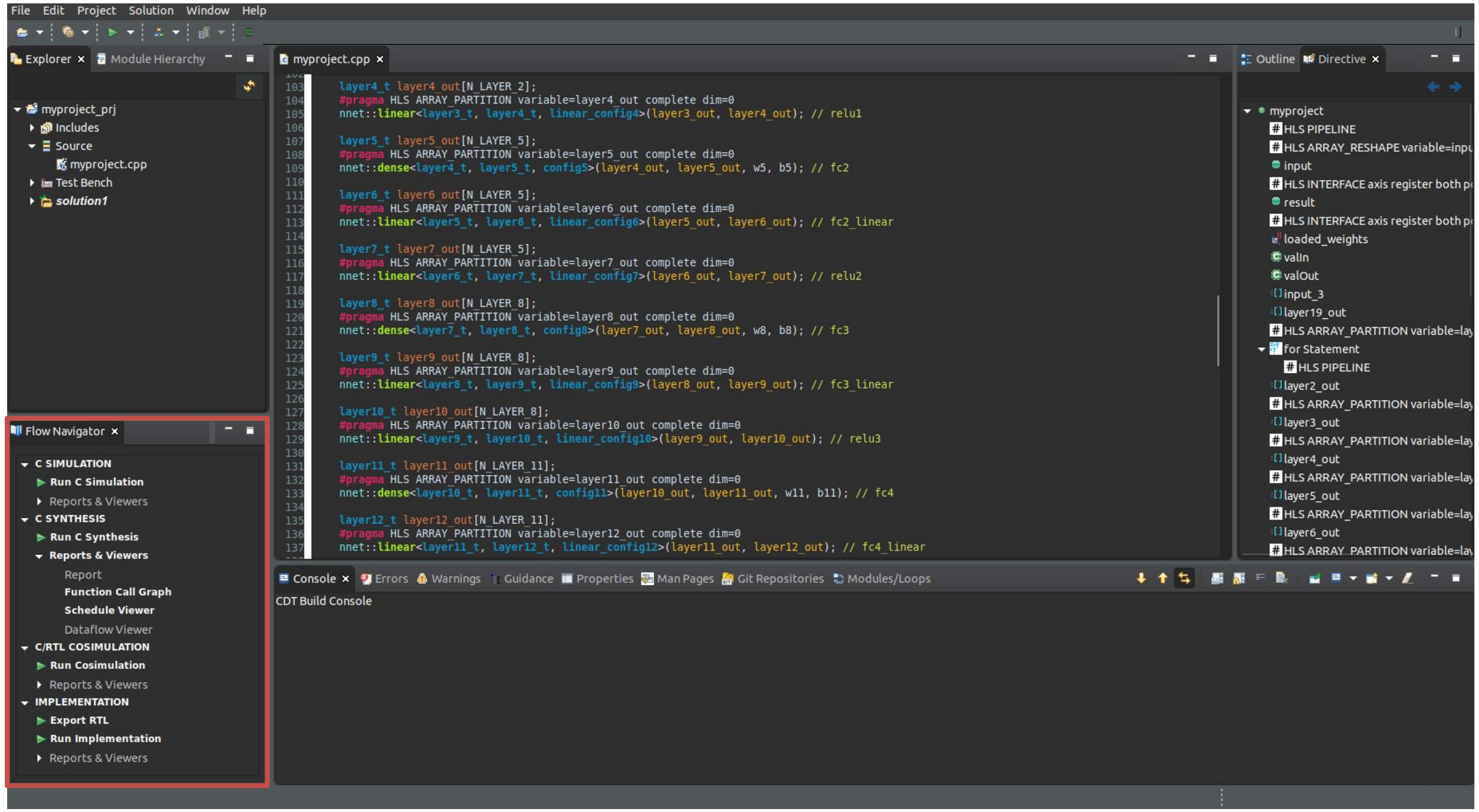
Vitis HLS GUI

Vitis HLS GUI



```
    layer4_t layer4_out[N_LAYER_2];
    #pragma HLS ARRAY_PARTITION variable=layer4_out complete dim=0
    nnet::linear<layer3_t, layer4_t, linear_config4>(layer3_out, layer4_out); // relu1

    layer5_t layer5_out[N_LAYER_5];
    #pragma HLS ARRAY_PARTITION variable=layer5_out complete dim=0
    nnet::dense<layer4_t, layer5_t, config5>(layer4_out, layer5_out, w5, b5); // fc2

    layer6_t layer6_out[N_LAYER_5];
    #pragma HLS ARRAY_PARTITION variable=layer6_out complete dim=0
    nnet::linear<layer5_t, layer6_t, linear_config6>(layer5_out, layer6_out); // fc2_linear

    layer7_t layer7_out[N_LAYER_5];
    #pragma HLS ARRAY_PARTITION variable=layer7_out complete dim=0
    nnet::linear<layer6_t, layer7_t, linear_config7>(layer6_out, layer7_out); // relu2

    layer8_t layer8_out[N_LAYER_8];
    #pragma HLS ARRAY_PARTITION variable=layer8_out complete dim=0
    nnet::dense<layer7_t, layer8_t, config8>(layer7_out, layer8_out, w8, b8); // fc3

    layer9_t layer9_out[N_LAYER_8];
    #pragma HLS ARRAY_PARTITION variable=layer9_out complete dim=0
    nnet::linear<layer8_t, layer9_t, linear_config9>(layer8_out, layer9_out); // fc3_linear

    layer10_t layer10_out[N_LAYER_8];
    #pragma HLS ARRAY_PARTITION variable=layer10_out complete dim=0
    nnet::linear<layer9_t, layer10_t, linear_config10>(layer9_out, layer10_out); // relu3

    layer11_t layer11_out[N_LAYER_11];
    #pragma HLS ARRAY_PARTITION variable=layer11_out complete dim=0
    nnet::dense<layer10_t, layer11_t, config11>(layer10_out, layer11_out, w11, b11); // fc4

    layer12_t layer12_out[N_LAYER_11];
    #pragma HLS ARRAY_PARTITION variable=layer12_out complete dim=0
    nnet::linear<layer11_t, layer12_t, linear_config12>(layer11_out, layer12_out); // fc4_linear
```

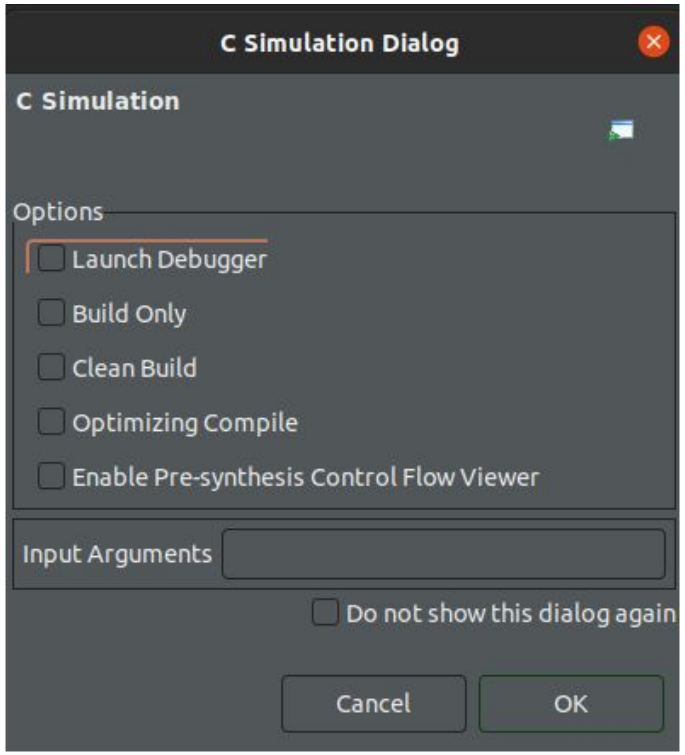
Vitis HLS GUI



```
    layer4_t layer4_out[N_LAYER_2];
    #pragma HLS ARRAY_PARTITION variable=layer4_out complete dim=0
    nnet::linear<layer3_t, layer4_t, linear_config4>(layer3_out, layer4_out); // relu1

    layer5_t layer5_out[N_LAYER_5];
    #pragma HLS ARRAY_PARTITION variable=layer5_out complete dim=0
    nnet::dense<layer4_t, layer5_t, config5>(layer4_out, layer5_out, w5, b5); // fc2

    layer6_t layer6_out[N_LAYER_5];
    #pragma HLS ARRAY_PARTITION variable=layer6_out complete dim=0
    nnet::linear<layer5_t, layer6_t, linear_config6>(layer5_out, layer6_out); // fc2_linear

    layer7_t layer7_out[N_LAYER_5];
    #pragma HLS ARRAY_PARTITION variable=layer7_out complete dim=0
    nnet::linear<layer6_t, layer7_t, linear_config7>(layer6_out, layer7_out); // relu2

    layer8_t layer8_out[N_LAYER_8];
    #pragma HLS ARRAY_PARTITION variable=layer8_out complete dim=0
    nnet::dense<layer7_t, layer8_t, config8>(layer7_out, layer8_out, w8, b8); // fc3

    layer9_t layer9_out[N_LAYER_8];
    #pragma HLS ARRAY_PARTITION variable=layer9_out complete dim=0
    nnet::linear<layer8_t, layer9_t, linear_config9>(layer8_out, layer9_out); // fc3_linear

    layer10_t layer10_out[N_LAYER_8];
    #pragma HLS ARRAY_PARTITION variable=layer10_out complete dim=0
    nnet::linear<layer9_t, layer10_t, linear_config10>(layer9_out, layer10_out); // relu3

    layer11_t layer11_out[N_LAYER_11];
    #pragma HLS ARRAY_PARTITION variable=layer11_out complete dim=0
    nnet::dense<layer10_t, layer11_t, config11>(layer10_out, layer11_out, w11, b11); // fc4

    layer12_t layer12_out[N_LAYER_11];
    #pragma HLS ARRAY_PARTITION variable=layer12_out complete dim=0
    nnet::linear<layer11_t, layer12_t, linear_config12>(layer11_out, layer12_out); // fc4_linear
```

```
CDT Build Console
```

Vitis HLS GUI



C Simulation Dialog

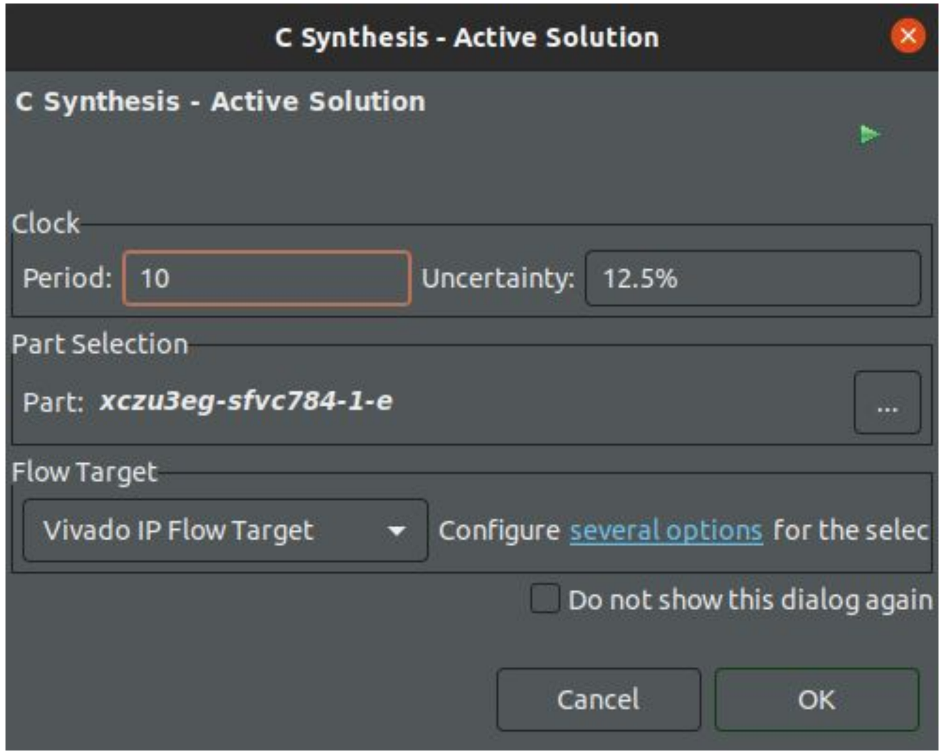
C Simulation

Options

- [ ] Launch Debugger
- [ ] Build Only
- [ ] Clean Build
- [ ] Optimizing Compile
- [ ] Enable Pre-synthesis Control Flow Viewer

Input Arguments

- [ ] Do not show this dialog again



C Synthesis - Active Solution

C Synthesis - Active Solution

Clock

Period: 10 Uncertainty: 12.5%

Part Selection

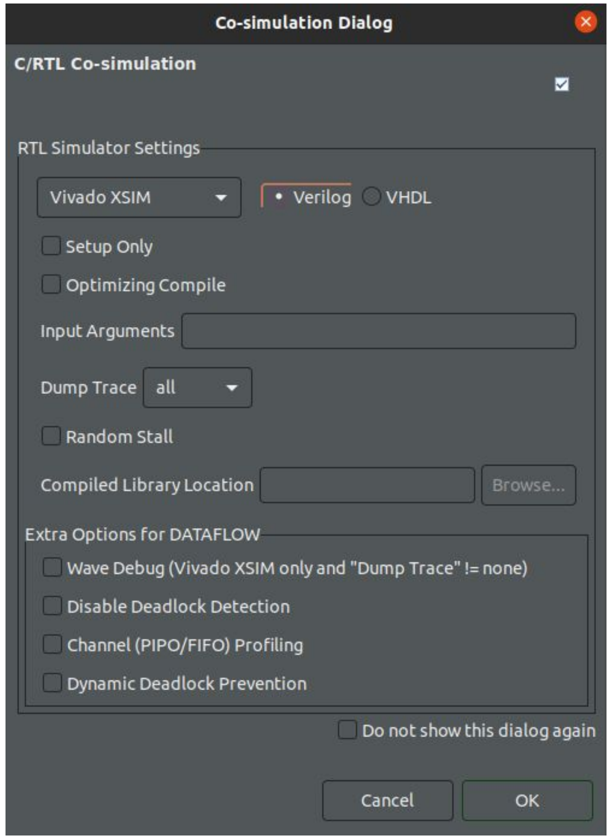
Part: *xczu3eg-sfvc784-1-e*

Flow Target

Vivado IP Flow Target Configure several options for the selec

- [ ] Do not show this dialog again

Vitis HLS GUI



Co-simulation Dialog

C/RTL Co-simulation

RTL Simulator Settings

Vivado XSIM [✓ Verilog] VHDL

- [ ] Setup Only
- [ ] Optimizing Compile

Input Arguments

Dump Trace: all

- [ ] Random Stall

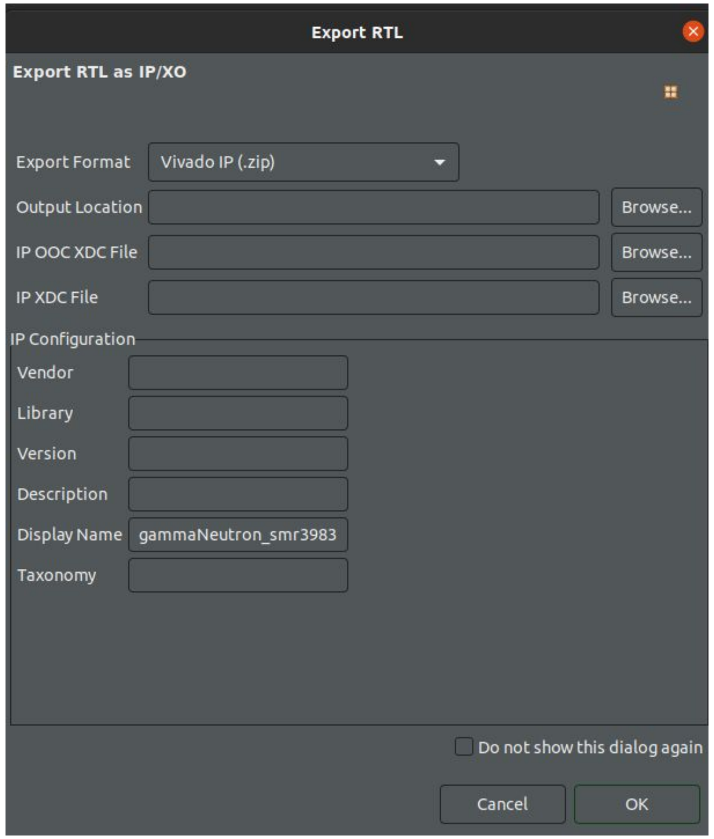
Compiled Library Location — Browse...

Extra Options for DATAFLOW

- [ ] Wave Debug (Vivado XSIM only and "Dump Trace" != none)
- [ ] Disable Deadlock Detection
- [ ] Channel (PIPO/FIFO) Profiling
- [ ] Dynamic Deadlock Prevention

- [ ] Do not show this dialog again

Cancel OK



Export RTL

Export RTL as IP/XO

Export Format	Vivado IP (.zip)
Output Location	
IP OOC XDC File	
IP XDC File	

IP Configuration

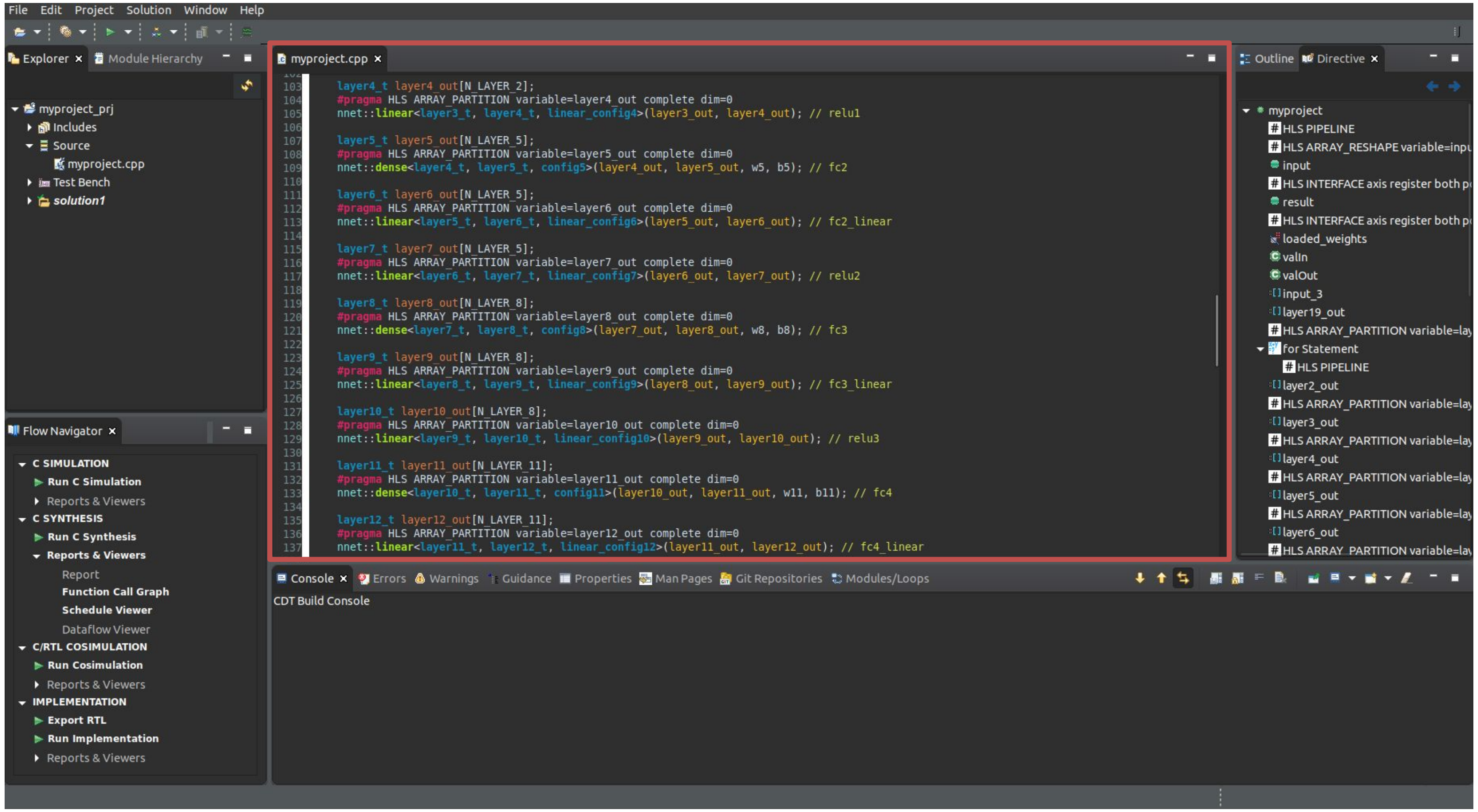
Vendor	
Library	
Version	
Description	
Display Name	gammaNeutron_smr3983
Taxonomy	

- [ ] Do not show this dialog again

Cancel OK

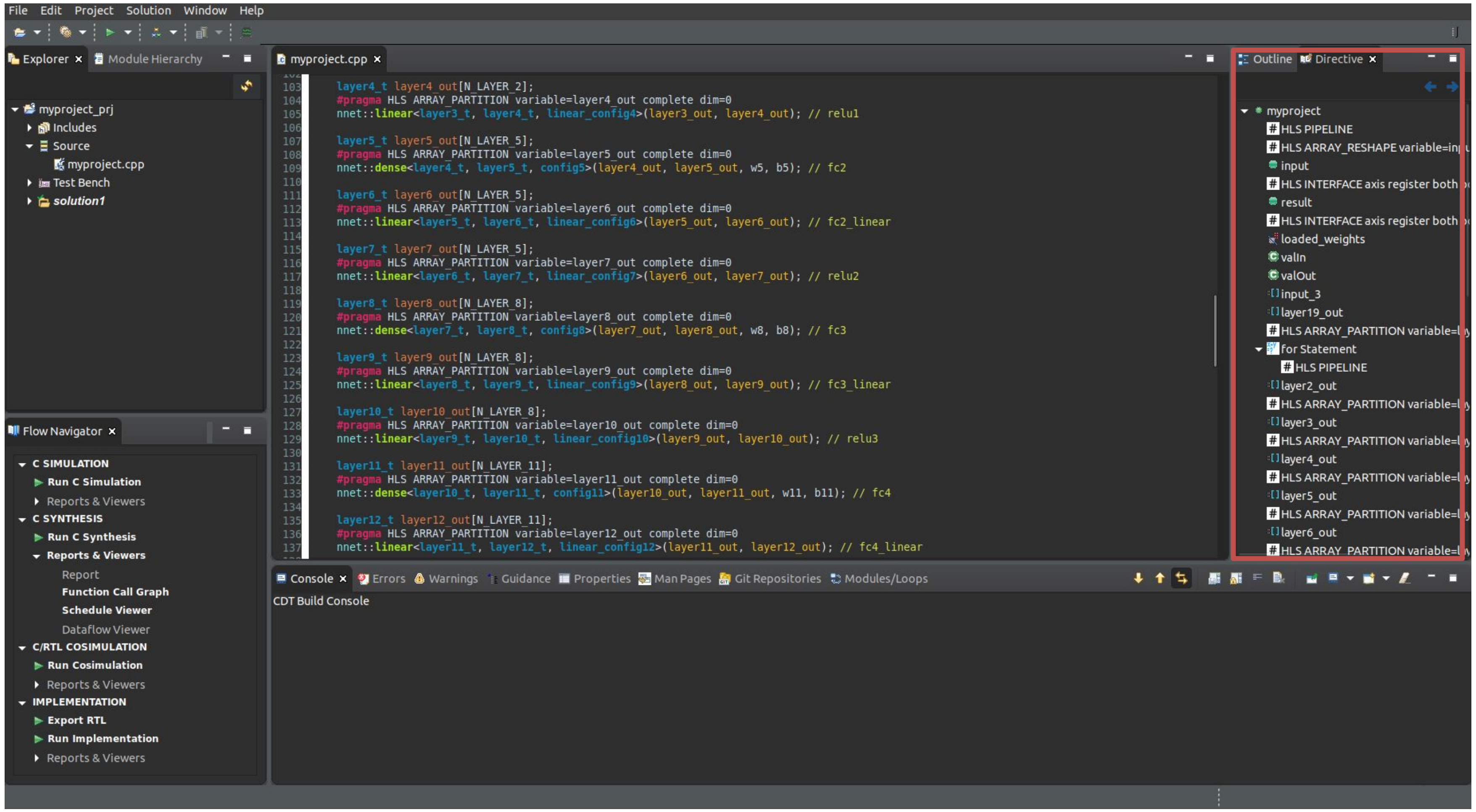
The Abdus Salam International Centre for Theoretical Physics

Vitis HLS GUI



```
    layer4_t layer4_out[N_LAYER_2];
    #pragma HLS ARRAY_PARTITION variable=layer4_out complete dim=0
    nnet::linear<layer3_t, layer4_t, linear_config4>(layer3_out, layer4_out); // relu1

    layer5_t layer5_out[N_LAYER_5];
    #pragma HLS ARRAY_PARTITION variable=layer5_out complete dim=0
    nnet::dense<layer4_t, layer5_t, config5>(layer4_out, layer5_out, w5, b5); // fc2

    layer6_t layer6_out[N_LAYER_5];
    #pragma HLS ARRAY_PARTITION variable=layer6_out complete dim=0
    nnet::linear<layer5_t, layer6_t, linear_config6>(layer5_out, layer6_out); // fc2_linear

    layer7_t layer7_out[N_LAYER_5];
    #pragma HLS ARRAY_PARTITION variable=layer7_out complete dim=0
    nnet::linear<layer6_t, layer7_t, linear_config7>(layer6_out, layer7_out); // relu2

    layer8_t layer8_out[N_LAYER_8];
    #pragma HLS ARRAY_PARTITION variable=layer8_out complete dim=0
    nnet::dense<layer7_t, layer8_t, config8>(layer7_out, layer8_out, w8, b8); // fc3

    layer9_t layer9_out[N_LAYER_8];
    #pragma HLS ARRAY_PARTITION variable=layer9_out complete dim=0
    nnet::linear<layer8_t, layer9_t, linear_config9>(layer8_out, layer9_out); // fc3_linear

    layer10_t layer10_out[N_LAYER_8];
    #pragma HLS ARRAY_PARTITION variable=layer10_out complete dim=0
    nnet::linear<layer9_t, layer10_t, linear_config10>(layer9_out, layer10_out); // relu3

    layer11_t layer11_out[N_LAYER_11];
    #pragma HLS ARRAY_PARTITION variable=layer11_out complete dim=0
    nnet::dense<layer10_t, layer11_t, config11>(layer10_out, layer11_out, w11, b11); // fc4

    layer12_t layer12_out[N_LAYER_11];
    #pragma HLS ARRAY_PARTITION variable=layer12_out complete dim=0
    nnet::linear<layer11_t, layer12_t, linear_config12>(layer11_out, layer12_out); // fc4_linear
```

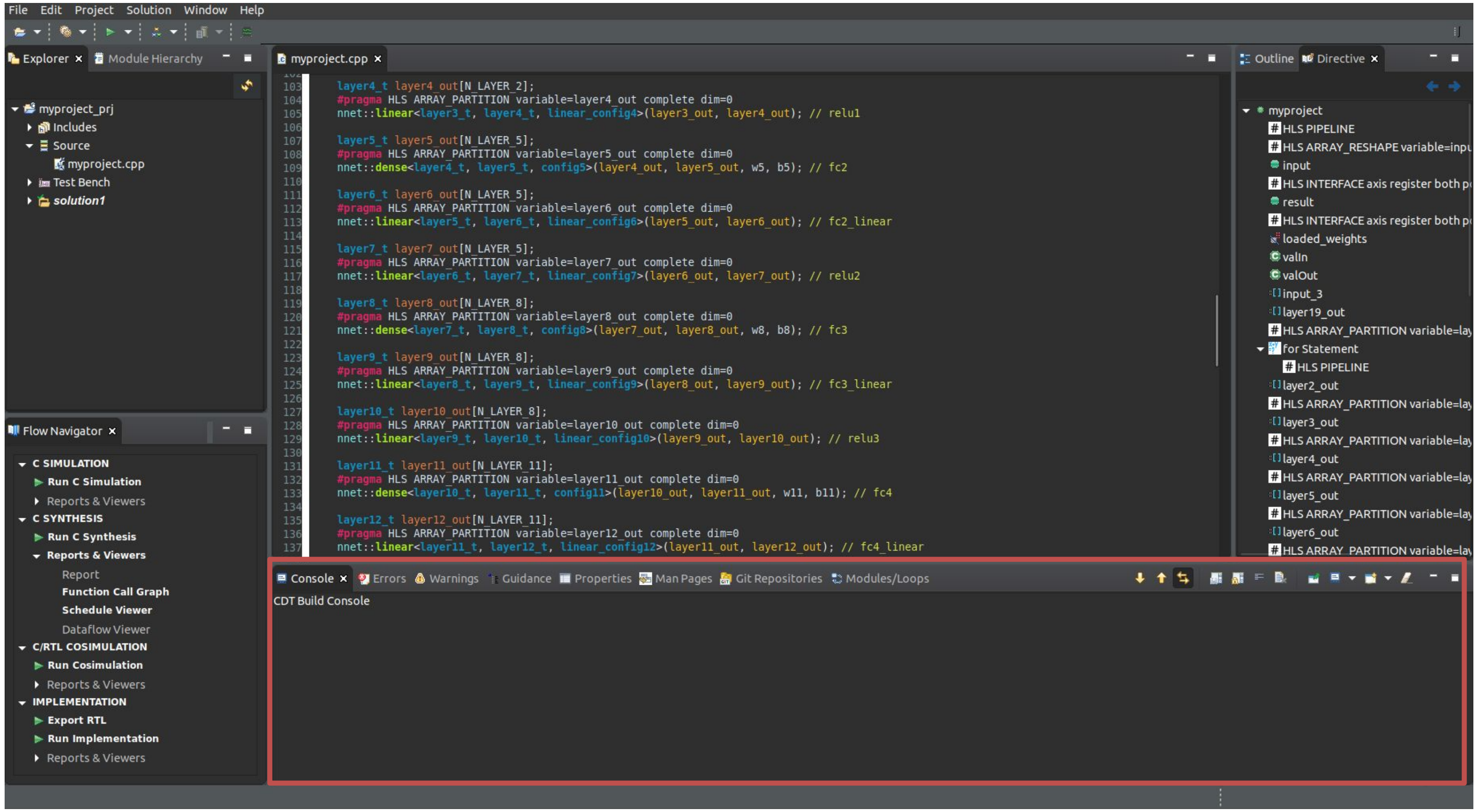
Vitis HLS GUI



```cpp
    layer4_t layer4_out[N_LAYER_2];
    #pragma HLS ARRAY_PARTITION variable=layer4_out complete dim=0
    nnet::linear<layer3_t, layer4_t, linear_config4>(layer3_out, layer4_out); // relu1

    layer5_t layer5_out[N_LAYER_5];
    #pragma HLS ARRAY_PARTITION variable=layer5_out complete dim=0
    nnet::dense<layer4_t, layer5_t, config5>(layer4_out, layer5_out, w5, b5); // fc2

    layer6_t layer6_out[N_LAYER_5];
    #pragma HLS ARRAY_PARTITION variable=layer6_out complete dim=0
    nnet::linear<layer5_t, layer6_t, linear_config6>(layer5_out, layer6_out); // fc2_linear

    layer7_t layer7_out[N_LAYER_5];
    #pragma HLS ARRAY_PARTITION variable=layer7_out complete dim=0
    nnet::linear<layer6_t, layer7_t, linear_config7>(layer6_out, layer7_out); // relu2

    layer8_t layer8_out[N_LAYER_8];
    #pragma HLS ARRAY_PARTITION variable=layer8_out complete dim=0
    nnet::dense<layer7_t, layer8_t, config8>(layer7_out, layer8_out, w8, b8); // fc3

    layer9_t layer9_out[N_LAYER_8];
    #pragma HLS ARRAY_PARTITION variable=layer9_out complete dim=0
    nnet::linear<layer8_t, layer9_t, linear_config9>(layer8_out, layer9_out); // fc3_linear

    layer10_t layer10_out[N_LAYER_8];
    #pragma HLS ARRAY_PARTITION variable=layer10_out complete dim=0
    nnet::linear<layer9_t, layer10_t, linear_config10>(layer9_out, layer10_out); // relu3

    layer11_t layer11_out[N_LAYER_11];
    #pragma HLS ARRAY_PARTITION variable=layer11_out complete dim=0
    nnet::dense<layer10_t, layer11_t, config11>(layer10_out, layer11_out, w11, b11); // fc4

    layer12_t layer12_out[N_LAYER_11];
    #pragma HLS ARRAY_PARTITION variable=layer12_out complete dim=0
    nnet::linear<layer11_t, layer12_t, linear_config12>(layer11_out, layer12_out); // fc4_linear
```

```
CDT Build Console
```

Vitis HLS GUI

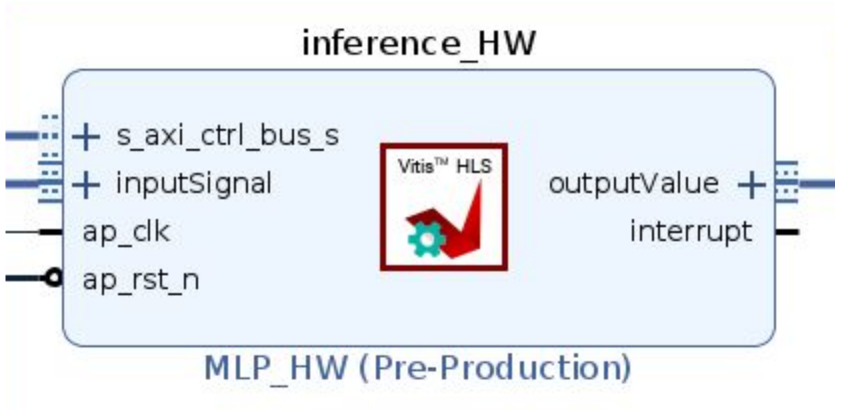


```
    layer4_t layer4_out[N_LAYER_2];
    #pragma HLS ARRAY_PARTITION variable=layer4_out complete dim=0
    nnet::linear<layer3_t, layer4_t, linear_config4>(layer3_out, layer4_out); // relu1

    layer5_t layer5_out[N_LAYER_5];
    #pragma HLS ARRAY_PARTITION variable=layer5_out complete dim=0
    nnet::dense<layer4_t, layer5_t, config5>(layer4_out, layer5_out, w5, b5); // fc2

    layer6_t layer6_out[N_LAYER_5];
    #pragma HLS ARRAY_PARTITION variable=layer6_out complete dim=0
    nnet::linear<layer5_t, layer6_t, linear_config6>(layer5_out, layer6_out); // fc2_linear

    layer7_t layer7_out[N_LAYER_5];
    #pragma HLS ARRAY_PARTITION variable=layer7_out complete dim=0
    nnet::linear<layer6_t, layer7_t, linear_config7>(layer6_out, layer7_out); // relu2

    layer8_t layer8_out[N_LAYER_8];
    #pragma HLS ARRAY_PARTITION variable=layer8_out complete dim=0
    nnet::dense<layer7_t, layer8_t, config8>(layer7_out, layer8_out, w8, b8); // fc3

    layer9_t layer9_out[N_LAYER_8];
    #pragma HLS ARRAY_PARTITION variable=layer9_out complete dim=0
    nnet::linear<layer8_t, layer9_t, linear_config9>(layer8_out, layer9_out); // fc3_linear

    layer10_t layer10_out[N_LAYER_8];
    #pragma HLS ARRAY_PARTITION variable=layer10_out complete dim=0
    nnet::linear<layer9_t, layer10_t, linear_config10>(layer9_out, layer10_out); // relu3

    layer11_t layer11_out[N_LAYER_11];
    #pragma HLS ARRAY_PARTITION variable=layer11_out complete dim=0
    nnet::dense<layer10_t, layer11_t, config11>(layer10_out, layer11_out, w11, b11); // fc4

    layer12_t layer12_out[N_LAYER_11];
    #pragma HLS ARRAY_PARTITION variable=layer12_out complete dim=0
    nnet::linear<layer11_t, layer12_t, linear_config12>(layer11_out, layer12_out); // fc4_linear
```

```
CDT Build Console
```

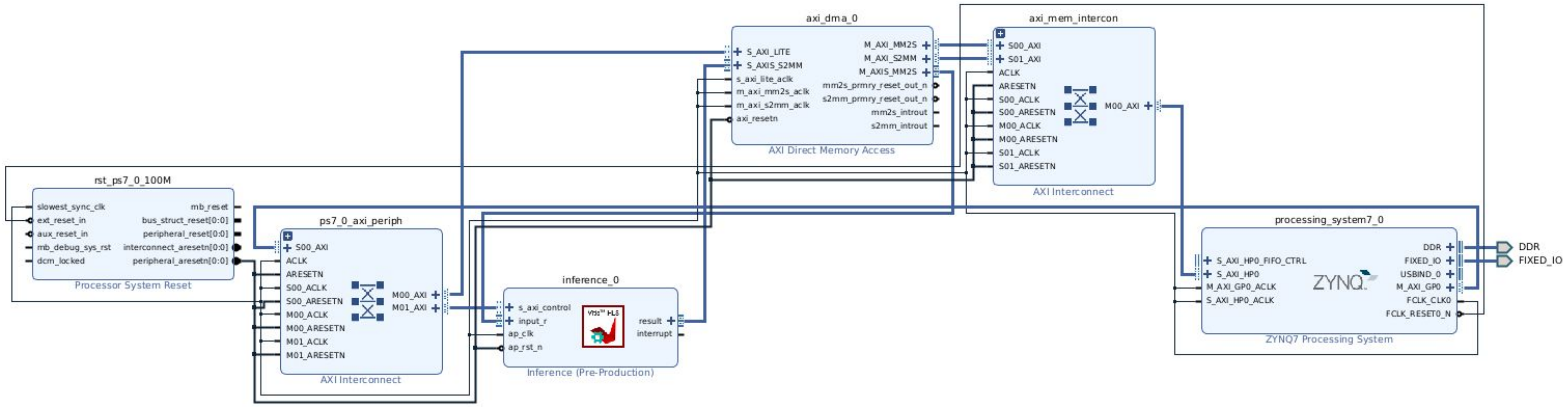
Vitis HLS GUI

Generated IP core



Vivado IP integration

Vivado IP integration



Demo HLS:
Vector Addition and Matrix Multiplication

The Abdus Salam International Centre for Theoretical Physics

High-Level Synthesis: Bridging Software and Hardware

Romina Soledad Molina, Ph.D.
MLab-STI, ICTP

Perú - Online - 2025 -

Universidad Tecnológica del Perú