Synthesis of Digital Systems
Lab Lecture -2

Getting started with Vivado-HLS

Munish Jassi

Recap from Lab-A

Learnings from Lab-A

- Understanding of Zedboard for SoC prototyping.
- Xilinx Tool chain for working with FPGA (PlanAhead, XPS, SDK).
- Setup for Programmable System (PS) with instantiation of peripherals.
- Instantiation of hardware IP (LED_controller) for Programmable Logic (PL).
- UCF design constraints and FPGA pin mapping.
- Software code for LED application and cross-compilation for ARM.
- Implementing the Hardware (.bit) and Software (.elf) on FPGA.
- Hardware interface to CPU via UART.

Other Topics from Lab-A

- What are external ports?
- Three booting modes available on ZedBoard.
- Important configuration files, MHS, MSS, XML, xparameters.h files.
- More on driver files and library generation.
- Inbyte() and Xgpio_DiscreteWrite() functions.
- minicom serial interface – baud rates and parity
- Xgpio_Initialize

Queries about Lab-A

- How many hours did you spend for implementation?

Content of the Lab Course

- System prototyping on FPGA and associated software tool chain.
- **Industry standard High Level Synthesis (HLS) tool chain.**
- Implementation of video processing hardware System and software interface.
- Hardware Accelerator (HA) for performance improvement.

High Level Synthesis

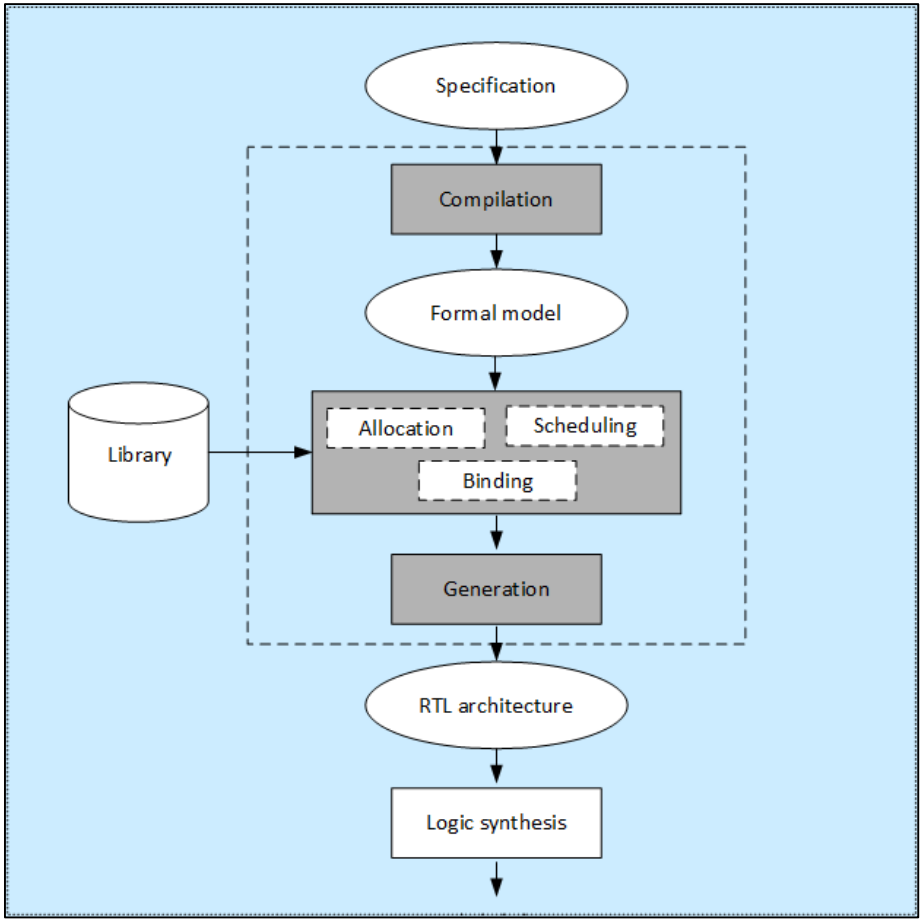
- Automated process
- High level abstraction for system design
- RTL generation

Input:	**Output**:
• Design specification and Constraints • Optimization directives • Module library of RTL components	• RTL implementation structure • Other features

- Automated generation of hardware IP from high level languages.
- Facilicate IP-reuse and Model-Based SoC design.
- Rapid prototyping and early time-to-market.

Steps for HLS

- Compilation and modelling
- Allocation
- Scheduling
- Binding
- Generation



Source: S. Katkoori, “High Level Synthesis and Reconfigurable Computing,” CSE Dept., Univ. of South Florida Design Automation Summer School June, 2011

HLS Tools

- High level system description languages
- Describe design parameters
- **Optimize source code**
- Generate RTL
- Various features

Optimization Techniques

- There are many Optimization techniques:
  - Loops (merging, unrolling, and pipelining)
  - Data arrays mapping (partitioning, reshaping and merging)
  - Function Calls

Optimization Techniques cont.

- Loops Optimization techniques:
  - Merging

```
void top (a[4],b[4],c[4],d[4],e[4],f[4].....) {
    ....
    Add: for (i=3 ; i>=0 ; i--)
    {
        if(x[i])
            a[i]=b[i] + c[i];
    }
    Sub: for (i=3 ; i>=0 ; i--)
    {
        if(!x[i])
            d[i]=e[i] - f[i];
    }
    ....
}
```

(A) Without Loop merging

```
void top (a[4],b[4],c[4],d[4],e[4],f[4].....) {
    ....

    Merge: for (i=3 ; i>=0 ; i--)
    {
        if(x[i])
            a[i]=b[i] + c[i];
        else
            d[i]=e[i] - f[i];
        ....
    }

    ....
}
```

(B) With Loop merging

  - Unrolling

```
void top (a[4],b[4],c[4].....) {
    ....

    for (i=0 ; i<N ; i++)
    {
        a[i]=b[i] + c[i];
    }
    ....
}
```

(A) Without Loop unrolling

```
void top (a[4],b[4],c[4].....) {
    ....
    for (i=0 ; i<N ; I+=2)
    {
        a[i]=b[i] + c[i];
        if (i+1 >= N) break;
        a[i+1]=b[i+1] + c[i+1];
    }
}
```

(B) With Loop unrolling

Optimization Techniques cont.

- Data Arrays Optimization techniques:
  - Partitioning
  - Reshaping
  - Merging/Mapping

Optimization Techniques cont.

- Function Call Optimization techniques:
  - Instantiation

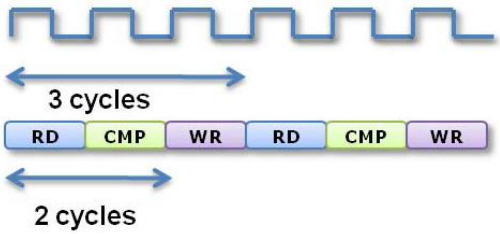
```
void A(){
  B(true);
  B(false);
  B(true);
  B(false);
}

void B(bool mode){
  if (mode) {
     // code segment 1
  } else {
     // code segment 2
  }
}
```

(A) Without Function Instantiation

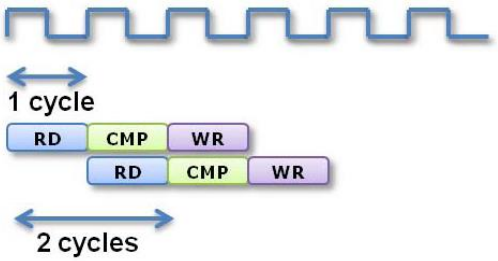
```
void A(){
  B1();
  B2();
  B1();
  B2();
}

void B1() {
  // code segment 1
}

void B2() {
  // code segment 2
}
```

(B) With Function Instantiation

  - Function Pipelining



(A) Without Function Pipelining



(B) With Function Pipelining

Source: Xilinx, “Vivado Design Suite User Guide,” www.xilinx.com, vol. 2012.2, 2012.

Overview of HLS Tools

- There are many HLS tools available.
  - Catapult C by Calypto
  - Vivado by Xilinx
  - C-to-Silicon by Cadence Design Systems
  - MDWorkbench by Sodius Corp
  - Kactus2 from Tampere University of Technology.
  - LegUp from University of Toronto
  - GAUT from Université de Bretagne Sud

- We will focus on Xilinx‘s Vivado-HLS tool (v2013.3).

Xilinx Vivado HLS

- Vivado-HLS
  - Input in C, C++, SystemC, Matlab and VHDL
  - Optimization loop directives, data arrays, function calls
  - Allocation of resources, scheduling, binding
  - RTL generation
  - Verification
  - Documentation and analysis

Source: http://www.xilinx.com/products/design-tools/vivado/

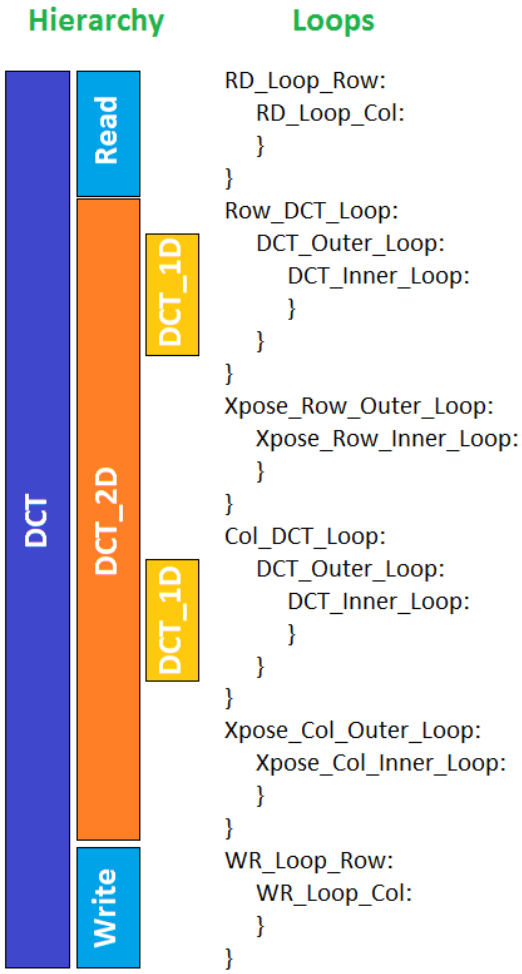
Description for Lab-B

Target Application: Discrete Cosine Transform (DCT)

- DCT of a finite sequence of data points is the sum of cosines at different frequencies.

$$X_k = \sum_{n=0}^{N-1} x_n \cos\left[\frac{\pi}{N}\left(n+\frac{1}{2}\right)k\right] \qquad k = 0, \ldots, N-1.$$

- DCT has computational complexity of $O(N^2)$
- DCT usage applies for Signal and image processing, data compression algorithms, JPEG, numerical solutions.

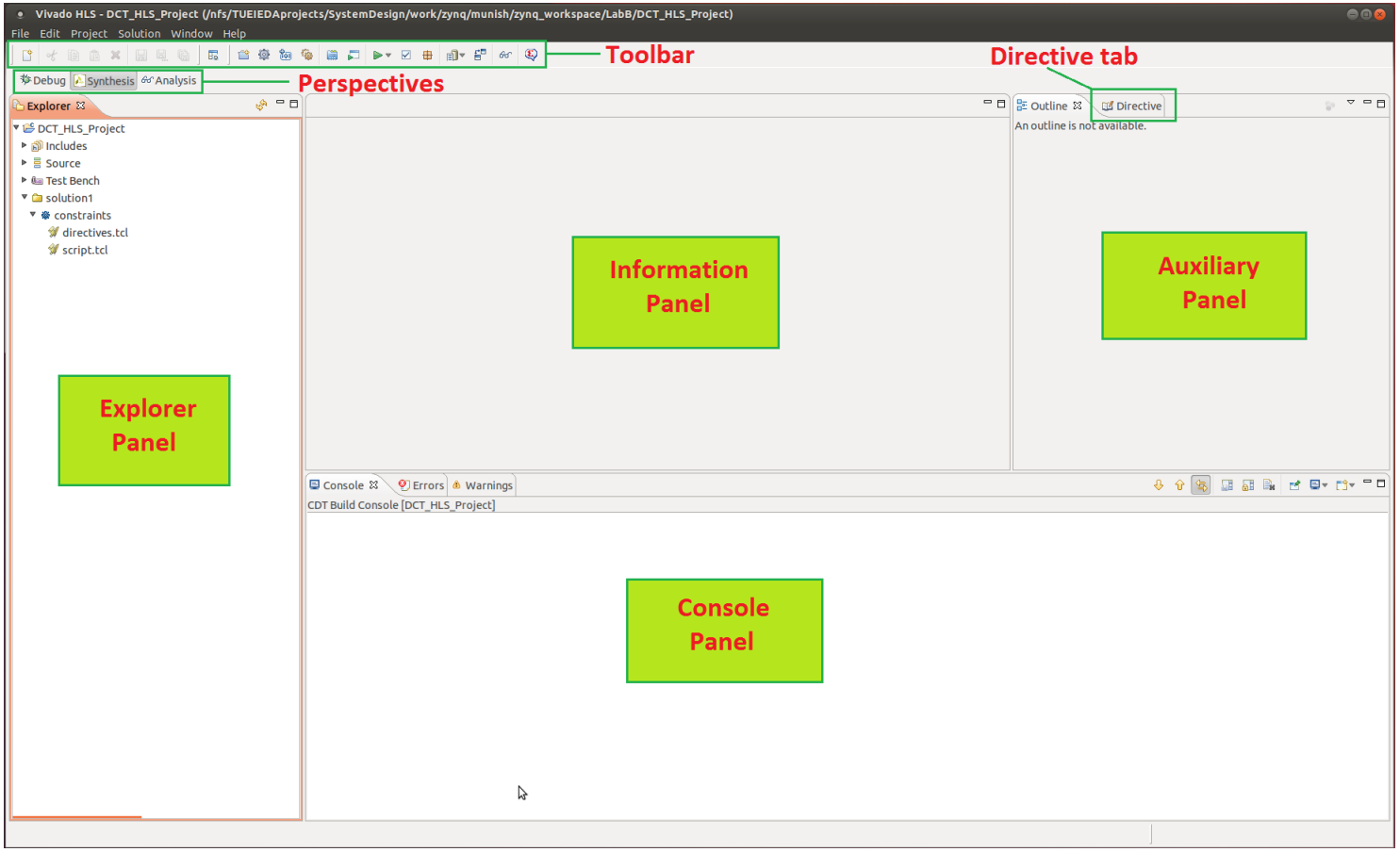


Targets for this Lab

- Target of this lab is to achieve throughput (interval) for DCT application of less than 300 clock cycles.

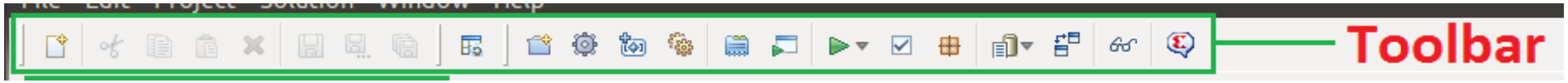
- Steps:
  1. Get familiar with HLS approach using Industrial Tool chain.
  2. Describe design inputs for the HLS.
  3. Analyze the synthesis results and use optimization directives to optimize the Design performances.
  4. Export the final IP core for EDK.

Vivado-HLS: Main panel

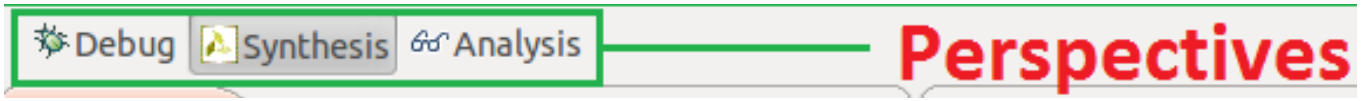


Vivado-HLS: Steps

- Index C Source
- Run C Simulation
- C Synthesis
- Run C/RTL Cosimulation
- Export RTL



Vivado-HLS: Perspectives



- Analysis Perspective

File Edit Project Solution Window Help

Debug Synthesis Analysis

Module Hierarchy

	BRAM	DSP	FF	LUT	Latency	Interval	Pipeline type
dct	6	1	183	355	3959	3960	none
dct_2d	4	1	135	263	3668	3668	none

Performance Profile | Resource Profile

	Pipelined	Latency	Initiation Interval	Iteration Latency	Trip count
dct	-	3959	3960	-	-
RD_Loop_Row	no	144	-	18	8
WR_Loop_Row	no	144	-	18	8

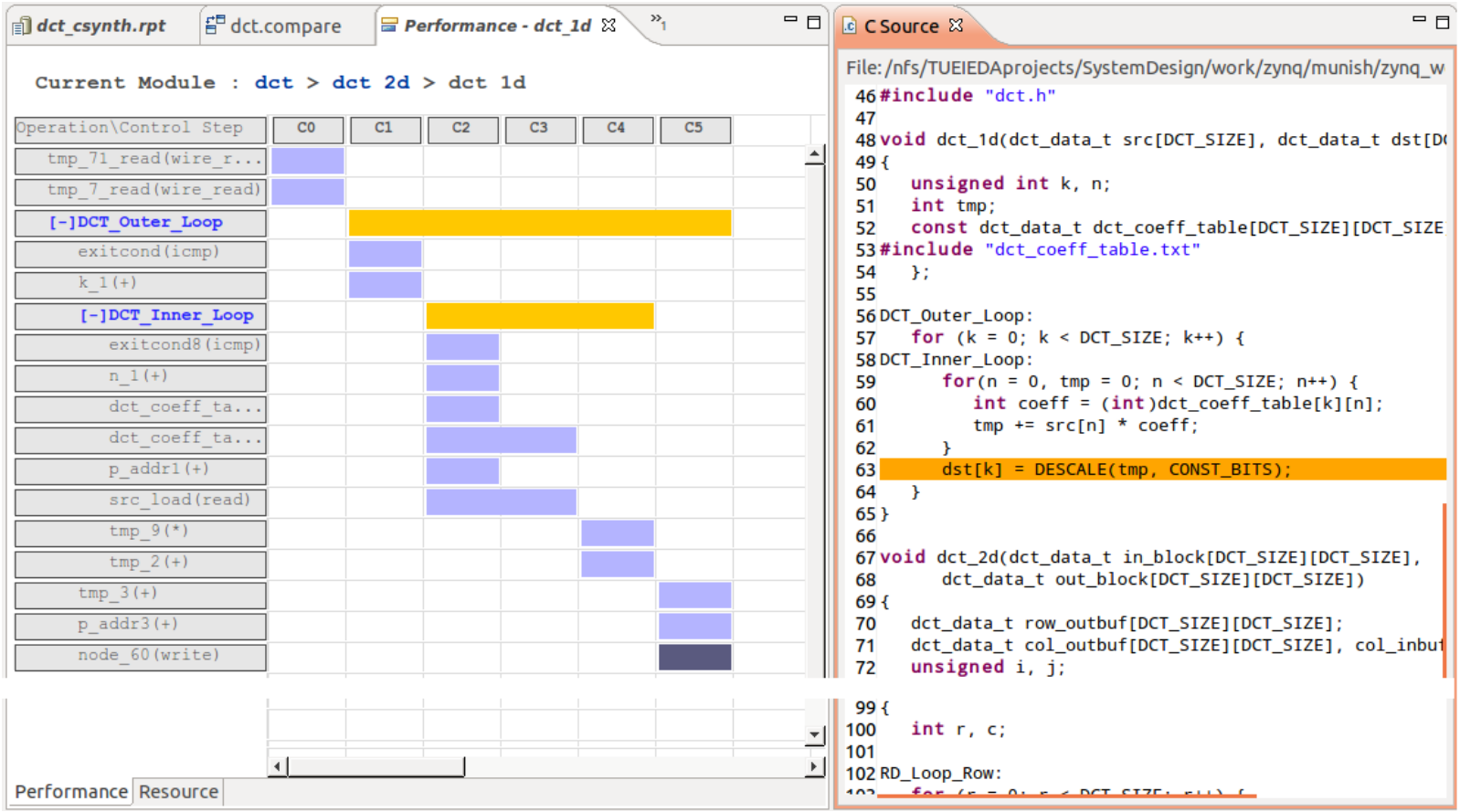
Performance - dct

Current Module : dct

Operation\Control Step	C0	C1	C2	C3	C4	C5
1-10 RD_Loop_Row						
11 dct_2d(function)						
12-21 WR_Loop_Row						

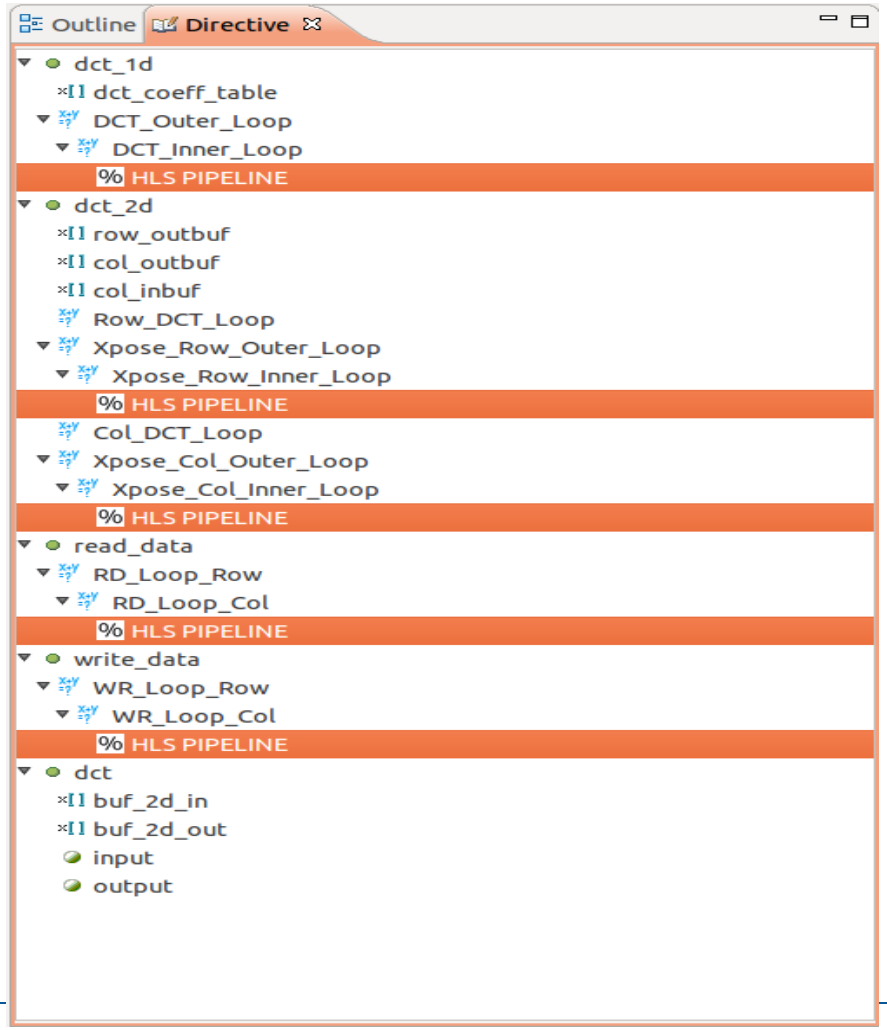
Performance | Resource

Vivado-HLS: Analysis Perspective



Vivado-HLS: Optimization Directives

- Directives are the optimization directions for the HLS tool.
- Directives could be applied to: Functions, Interfaces, Arrays, Loops, Regions.
- Directives :
  - Pipeline
  - Inline
  - Loop_Flatten
  - Datapath
  - and others



Vivado-HLS: IP Packaging

- **Export RTL** to export Pcore IP for EDK.
- **Pcore** is a Xilinx's format for IP Core, used for integration and management of IP cores for XPS projects.

- To use exported Pcores in the SoC,

  Copy <project_dir>/<Solution>/impl/pcores/* to <EDK_proj>/pcores
- Project > Rescan Local repository, to list IP block as *available IPs* in *EDK.*

Getting familiar with Xilinx Vivado-HLS

Get started with Vivado-HLS

1. Vivado-HLS, 2013.3 version
2. Copy getting started project from <project docs>/GettingStartedHLS to <work dir>
   - **<work dir>** : */usr/local/labs/SDS/current/<your_lrz_account>/*
   - **project docs:** */usr/local/labs/SDS/current/<your_lrz_account>/project_documents*
3. To load environmental settings, *$module load xilinx/vivado/2013.3*
4. To start tool, Vivado-HLS: *$vivado_hls –p fir_prj.*
   If you are not able to invoke vivado_hls_v2013.3, try directly invoking the binary as /nfs/tools/xilinx/Vivado_HLS/2013.3/bin/vivado_hls -p fir_prj . Get familiar with the Vivado-HLS options.
5. Click *C-Simulation*. It will end with Pass! message on console
6. Click *C-Synthesis*. Analysis the final synthesis report.
7. Click *Analysis perspective.* Get familiar with options and reports.

Thank You

for your participation