INTRODUCTION TO FPGA PROGRAMMING

Lesson 14: Advanced Vivado Usage

Dr. Davide Cieri[1]

[1]Max-Planck-Institut für Physik, Munich

September 2024

MAX-PLANCK-INSTITUT FÜR PHYSIK

REMINDER: VIVADO FLOW

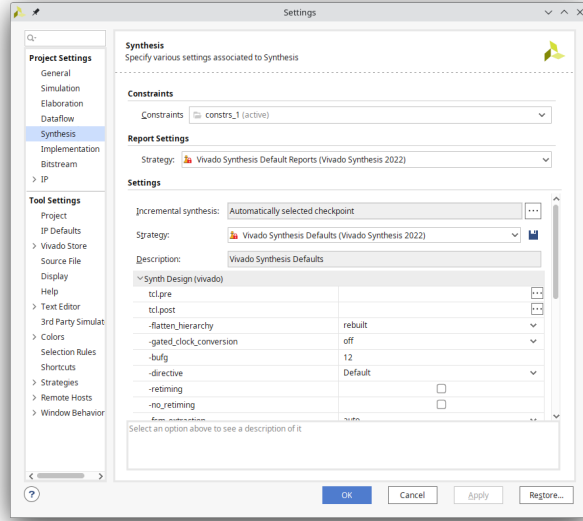
1. **HDL Design**
2. **Behavioural Simulation**
3. **Synthesis**
4. **Post-Synthesis Simulation**
5. **Implementation**
6. **Bitstream Generation**

VIVADO PROJECTS

- Vivado projects are device specific
  - Board or FPGA part (down to to the speed grade)
- The project include all the source file to compile your design
  - Design Sources (RTL, IPs), Simulation, Constraints
- Supported Languages in Vivado Synthesis 2022.2
  - SystemVerilog:(IEEE Std 1800-2012
  - Verilog: (IEEE Std 1364-2005)
  - VHDL (2008)

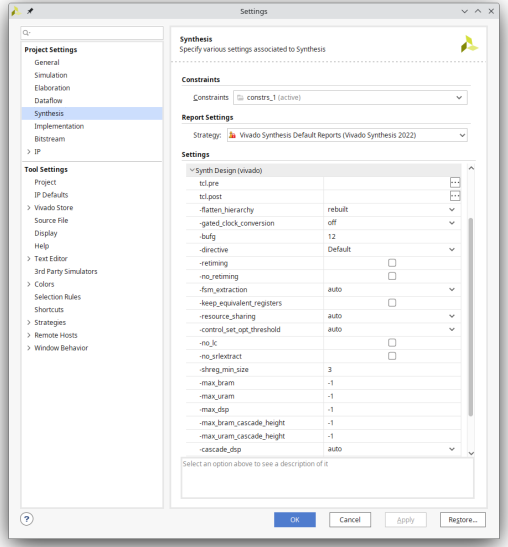
SYNTHESIS CONFIGURATION

- The synthesis process can be highly configured, by changing several Settings
- **Settings > Synthesis**
- Vivado provides several predefined strategies, for different applications
- Full list of settings here



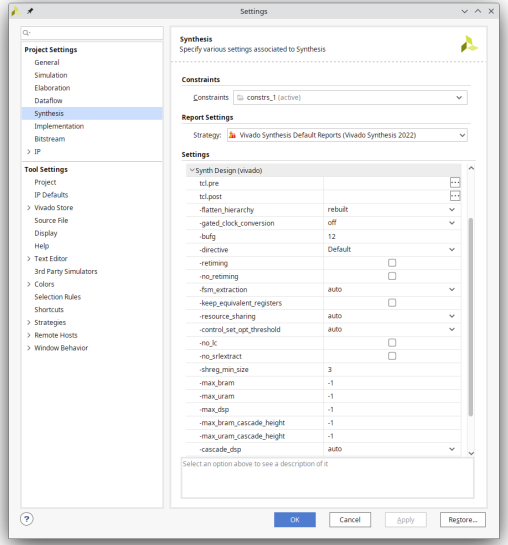
SYNTHESIS SETTINGS

- Most used synthesis settings:
  - `tcl.pre` and `tcl.post`: Run a custom tcl script before/after the synthesis
  - `flatten_hierarchy`: none (same hierarchy as the RTL), full (flats full hierarchy), rebuilt (flats the hierarchy and rebuilds it after synthesis)
  - `gated_clock_conversion`: allows synthesis of clocked logic with enables



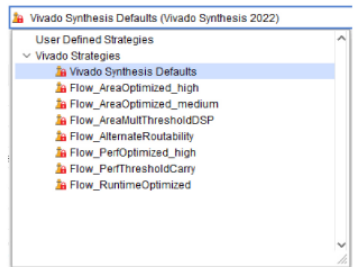
SYNTHESIS SETTINGS

- Most used synthesis settings:
  - `Directive`: Run synthesis with different optimisations
  - `bufg/max_bram/max_dsp`: Set a limit on the maximum resources to be used
  - `fsm_extraction`: Set the encoding scheme for all FSM in the design
  - `retiming`: Improve circuit timing performance



SYNTHESIS RUN STRATEGIES

- Vivado provides preconfigured run strategies, for different requirements
- You can also save the current synthesis settings in a *User Defined Strategy*
- This will be available in any Vivado project, on your pc
- Strategies can be also exported, and shared with colleagues



SYNTHESIS ATTRIBUTES

- We already met a few attributes that you can set inside your VHDL to force behaviours in the synthesis (e.g. `RAM_STYLE, FSM_ENCODING`)
- Attributes can be set inside your VHDL, but also in the XDC constraint file

VHDL Syntax

```
-- Syntax
attribute <ATTRIBUTE_NAME> : string;
attribute of <OBJECT> : signal is <VALUE>
-- Example
attribute ram_style : string;
attribute ram_style of myram : signal is "
    distributed";
```

XDC Syntax

```
-- Syntax
set_property <ATTRIBUTE_NAME> <
    ATTRIBUTE_VALUE> <OBJECT>;
-- Example
set_property ram_style "distributed" [
    get_cells myram]
```

AVAILABLE SYNTHESIS ATTRIBUTES

- Full List here
- `ASYNC_REG`: Informs the tool that a register is capable of receiving asynchronous data in the D input pin relative to the source clock (e.g. double-flopping synchroniser)
- `DONT_TOUCH`: Prevents the tool to optimise away a signal
- `MARK_DEBUG`: Add a signal for debug (creating an ILA)
- `USE_DSP`: Force the tool to place logic into a DSP or not
- `GATED_CLOCK`: To use in conjunction with the setting `gated_clock_conversion`. Informs the tool that the clock has a clock enable.

POST-SYNTHESIS SIMULATION

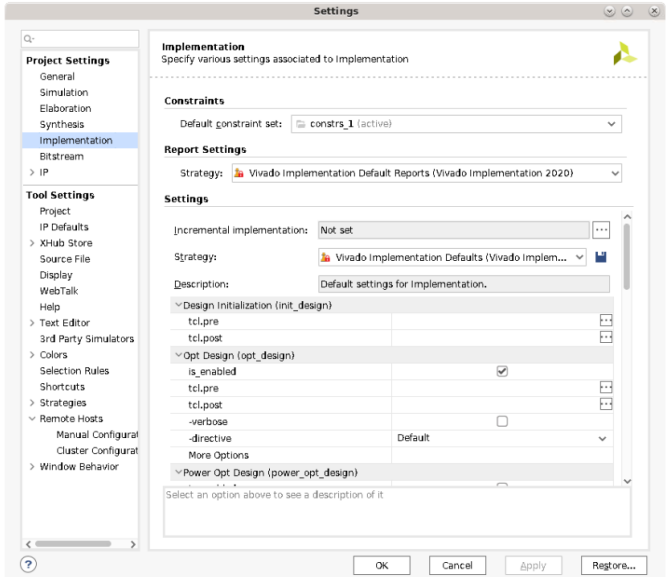
- Once our design has been synthesised, we know:
  - What logic blocks have been generated;
  - What delay each block adds.
- Two types of simulation can be run after synthesis:
  - **Post-Synthesis Functional Simulation**: faster, runs the simulation with the synthetised nets
  - **Post-Synthesis Timing Simulation**: Slower, add a 100ps delay to each storage element

VIVADO IMPLEMENTATION FLOW

- The Vivado implementation is divided in several steps:
  1. **Opt Design**: Optimises the logical design to make it easier to fit onto the FPGA;
  2. **Power Opt Design** (optional): Optmises the design to reduce power demands
  3. **Place Design**: Places the design onto FPGA, and replicates logic to improve timing
  4. **Post-Place Power Opt Design** (optional): Additional optimization to reduce power after placement.
  5. **Post-Place Phys Opt Design** (optional): Optimises logic and placement using estimated timing
  6. **Route Design**: Routes the design onto the FPGA
  7. **Post-Route Phys Opt Design** (optional): Optmises design using the actual routed delays

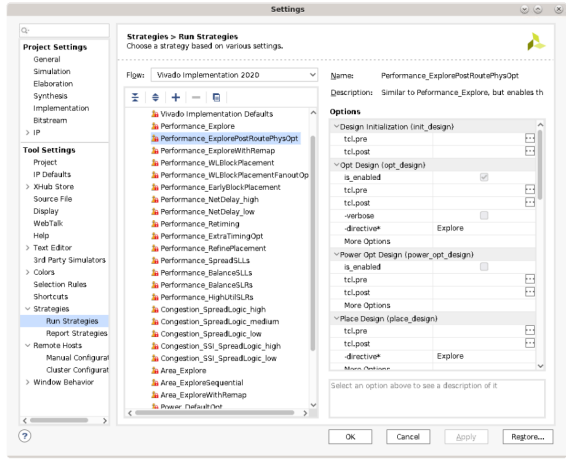
IMPLEMENTATION CONFIGURATION

- Also the implementation process can be configured
- **Settings > Implementation**
  - Each Implementation stage can be configured with different directives
  - Tcl scripts can be run before/after each step
- Full list of settings here

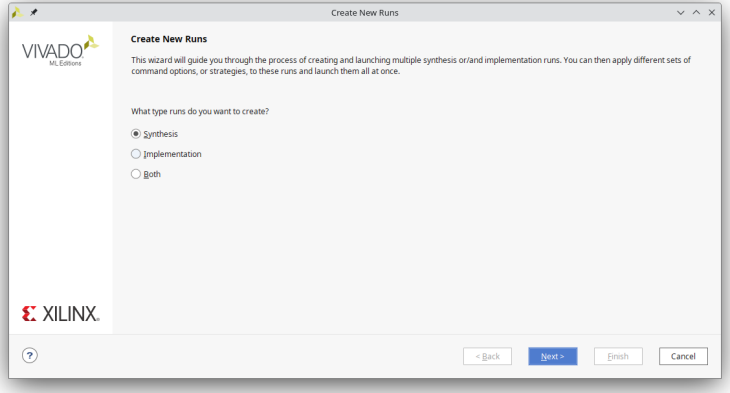


IMPLEMENTATION STRATEGIES

- Similarly to the synthesis, several pre-configured implementation strategies are available
- User settings can be saved in a user run strategies as well



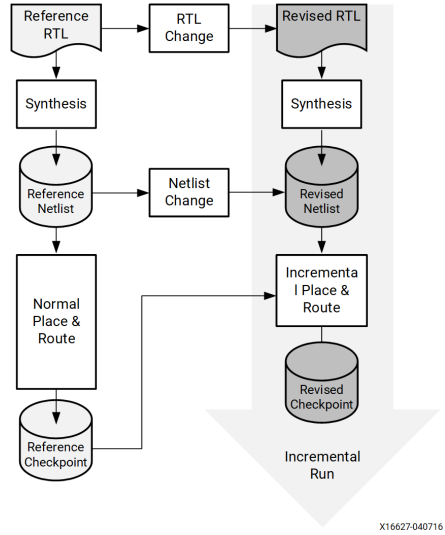
DESIGN RUNS



- Vivado allows you to create multiple design runs, to compare results of different strategies.
- Design runs can be created independently for synthesis and implementation

INCREMENTAL IMPLEMENTATION

- Large projects can take long time to implement (days)
- With incremental implementation, Place & Route starts from a reference checkpoint, before RTL changes
- Can speed up design implementation



SCRIPTING WITH VIVADO

- Vivado can also be run in batch mode
- Everything we did in this course can be done using only TCL scripts
- Every command you launch in the GUI is printed in the TCL console
- An Export feature allows you to export all command to recreate the project in a tcl script (File > Project > Write Tcl)

```
# Open a Vivado project in batch mode
vivado -mode tcl project.xpr
# Source a TCL script with Vivado in batch mode
vivado -mode batch -source mytclscript.tcl
```

TCL EXAMPLE

lab22.tcl

/home/dcieri/Work/fpga-course-tum/labs-solutions/lab22/sim/lab22.tcl

```
create_project ${_xil_proj_name_} ./${_xil_proj_name_} -part xc7a35tcpg236-1

# Set the directory path for the new project
set proj_dir [get_property directory [current_project]]

# Set project properties
set obj [current_project]
set_property -name "board_part_repo_paths" -value "[file normalize "$origin_dir/../../../../../.Xilinx/Vivado/2022.2/xhub/board_store/xilinx_board_stor
set_property -name "board_part" -value "digilentinc.com:basys3:part0:1.2" -objects $obj
set_property -name "default_lib" -value "xil_defaultlib" -objects $obj
set_property -name "enable_resource_estimation" -value "0" -objects $obj
set_property -name "enable_vhdl_2008" -value "1" -objects $obj
set_property -name "ip_cache_permissions" -value "read write" -objects $obj
set_property -name "ip_output_repo" -value "$proj_dir/${_xil_proj_name_}.cache/ip" -objects $obj
set_property -name "mem.enable_memory_map_generation" -value "1" -objects $obj
set_property -name "platform.board_id" -value "basys3" -objects $obj
set_property -name "revised_directory_structure" -value "1" -objects $obj
set_property -name "sim.central_dir" -value "$proj_dir/${_xil_proj_name_}.ip_user_files" -objects $obj
set_property -name "sim.ip.auto_export_scripts" -value "1" -objects $obj
set_property -name "simulator_language" -value "Mixed" -objects $obj
set_property -name "sim_compile_state" -value "1" -objects $obj
set_property -name "target_language" -value "VHDL" -objects $obj
set_property -name "webtalk.xsim_launch_sim" -value "6" -objects $obj

# Create 'sources_1' fileset (if not found)
if {[string equal [get_filesets -quiet sources_1] ""]} {
  create_fileset -srcset sources_1
}

# Set 'sources_1' fileset object
set obj [get_filesets sources_1]
set files [list \
 [file normalize "${origin_dir}/../lab22.srcs/sources_1/inports/src/LED_Counter.vhd"] \
]
add_files -norecurse -fileset $obj $files

# Set 'sources_1' fileset file properties for remote files
set file "$origin_dir/../lab22.srcs/sources_1/inports/src/LED_Counter.vhd"
set file [file normalize $file]
set file_obj [get_files -of_objects [get_filesets sources_1] [list "*$file"]]
```

USEFUL VIVADO TCL COMMNANDS

```
# Create the project
create_project -part <target_device> <project_name> <project_directory>
# Open Existing Project
open_project -file <project_file>
# Add files to synthesis fileset sources_1
add_files -fileset [get_filesets sources_1] <files>
# Add files to constraint fileset constrs_1
add_files -fileset [get_filesets constrs_1] <files>
# Add files to simulation fileset sim_1
add_files -fileset [get_filesets sim_1] <files>
# Launch Run (e.g. synth_1, impl_1)
launch_runs <run> -jobs <no. parallel jobs>
# Wait for run to finish
wait_on_run <run>
# Reset run
reset_run <run>
# Write Bitstream
launch_runs <implementation run> -to_step write_bitstream -jobs <no. parallel jobs>
```

USEFUL TCL DOCUMENTATION

- Tcl 8.6 Manual https://www.tcl.tk/man/tcl8.6/
- Vivado Design Suite Tcl Command Reference Guide (link)
- Most of the commands have a `help` flag. E.g. from the Vivado Tcl console

```
# Syntax
<command> -help
# Example
open_project -help
```

LAB 22: ADVANCED VIVADO FLOW

The figures in these slides are taken from:

- Digital Design: Principles and Practices, Fourth Edition, John F. Wakerly, ISBN 0-13- 186389-4.

- allaboutfpga.com
- nandland.com
- docs.amd.com
- https://www.symmetryelectronics.com/
- https://www.edn.com/
- Stephen A. Edwards, Columbia University, Fundamentals of Computer Systems, Spring 2012
- adafruit.com
- www.icdesigntips.com
- techdocs.altium.com
- anysilicon.com
- Yngve Hafting 2021, University of Oslo
- www.myomron.com