Workshop on
Fully Programmable
Systems-on-Chip for
Scientific Applications

Tool Command Language (TCL)

Luis Guillermo García Ordóñez

What is TCL?

TCL (Tool Command Language) or ***“Tickle”*** is a powerful, flexible scripting language initially developed in the late **1980**s by John Ousterhout. It's designed for command scripting and rapid prototyping.

- **Key Characteristics**:
    - **Interpreted Language**: TCL scripts are executed line-by-line, making it easy to test and modify in real-time.
    - **Cross-Platform Compatibility**: Runs on various operating systems, making it versatile across different development environments.
    - **Extensible**: TCL supports integration with other languages and tools, allowing it to adapt to various workflows.

Why TCL is relevant?

Widely Used in FPGA Tools: Major FPGA design tools, such as Xilinx Vivado, Intel Quartus, and Microsemi Libero, include TCL as their primary scripting language.

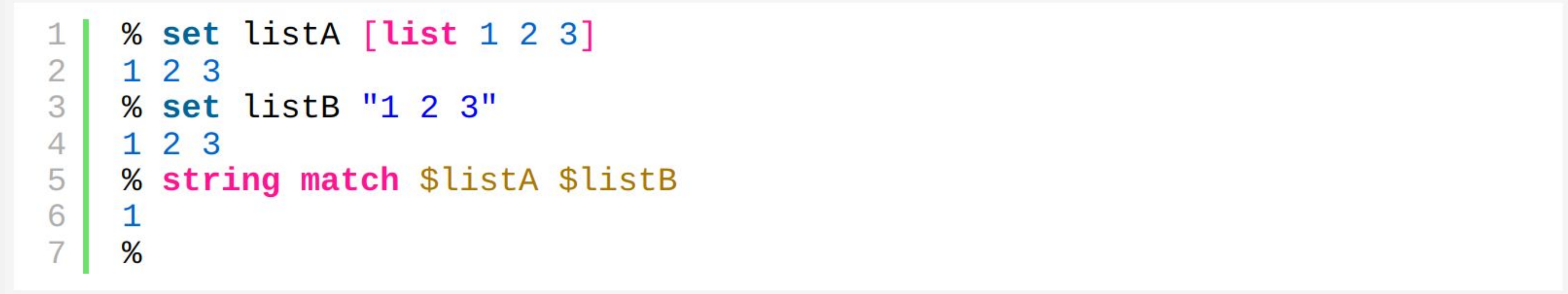
Automation and Control: TCL scripts can control FPGA design tools for tasks like synthesis, simulation, and implementation. It allows engineers to automate complex workflows that would be tedious and error-prone if done manually.

Adaptability: TCL provides a way to customize and streamline the FPGA design process, making it easier to adapt the workflow to specific project requirements and increase productivity.

TCL is both ingenious and frustrating

Tcl interpreters follow a basic set of rules, and that’s what makes it a good tool command language in the first place.

Everything is a string:



```
% set listA [list 1 2 3]
1 2 3
% set listB "1 2 3"
1 2 3
% string match $listA $listB
1
%
```

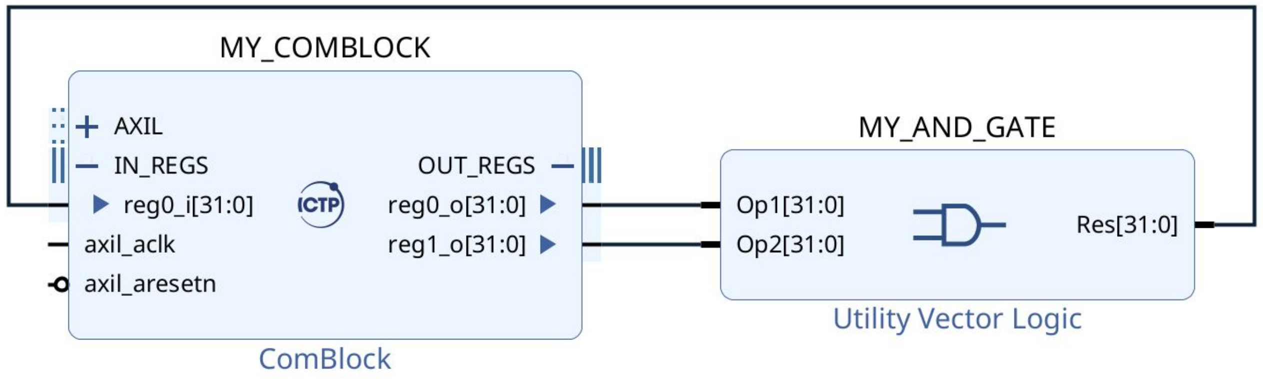
Everything can be redefined

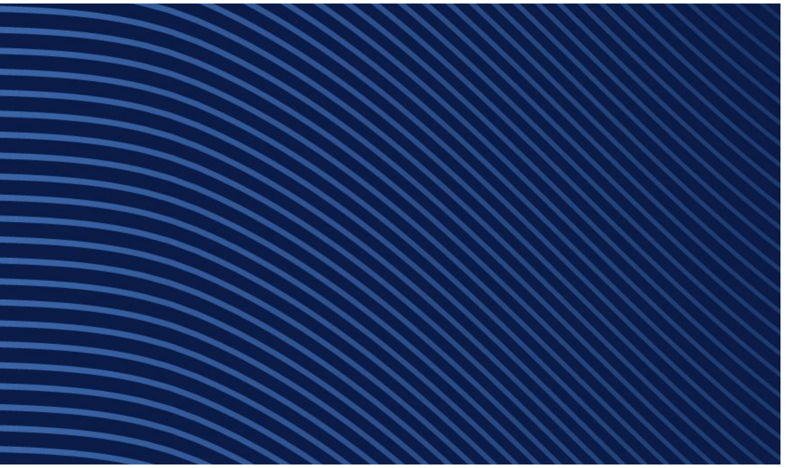
```
% puts "Hello World!"
Hello World!
%
% # Let's redefine puts to do something more
% rename puts puts_orig
% proc puts {args} {puts_orig "BMAZED! $args"}
%
% puts "Hello World!"
BMAZED! {Hello World!}
%
```

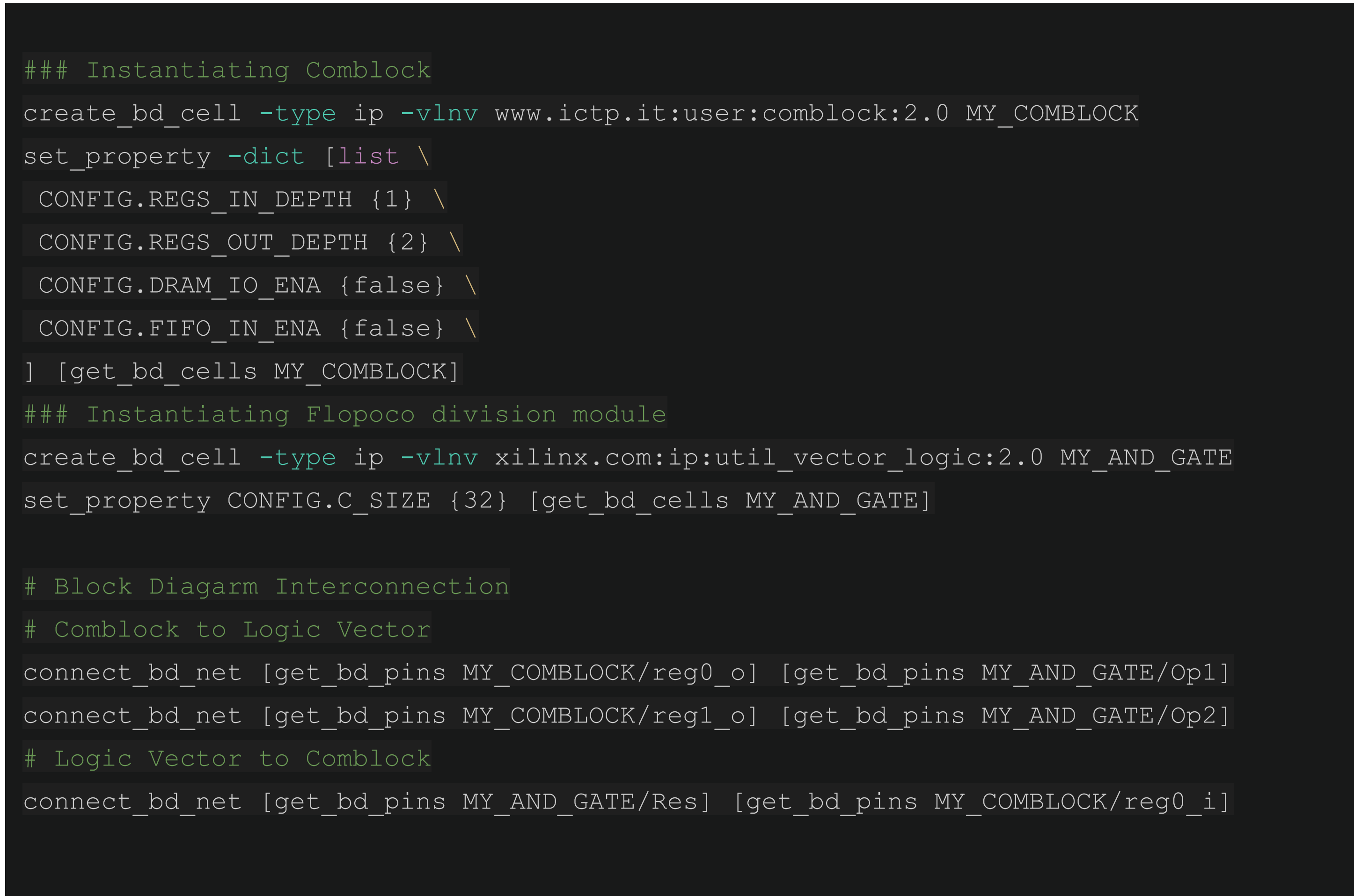
Role of TCL in FPGA Design Flow

- **Automating Repetitive Tasks:** SoC design involves many repetitive tasks, such as defining constraints, running synthesis, or generating reports. TCL scripts can be written to perform these tasks automatically. E.g. Synthesis, implementation, and generate a bitstream.
- **Configuration and Setup:**TCL scripts are frequently used to configure the design environment, including initializing settings and setting up design constraints, paths, and other variables. E.g. Board selection, Constraints (timing, placement, etc).
- **Synthesis, Simulation, and Implementation**
- **Batch Processing:** Multiple simulations, implementations, or tests need to be run on different configurations or parameters.

TCL Example



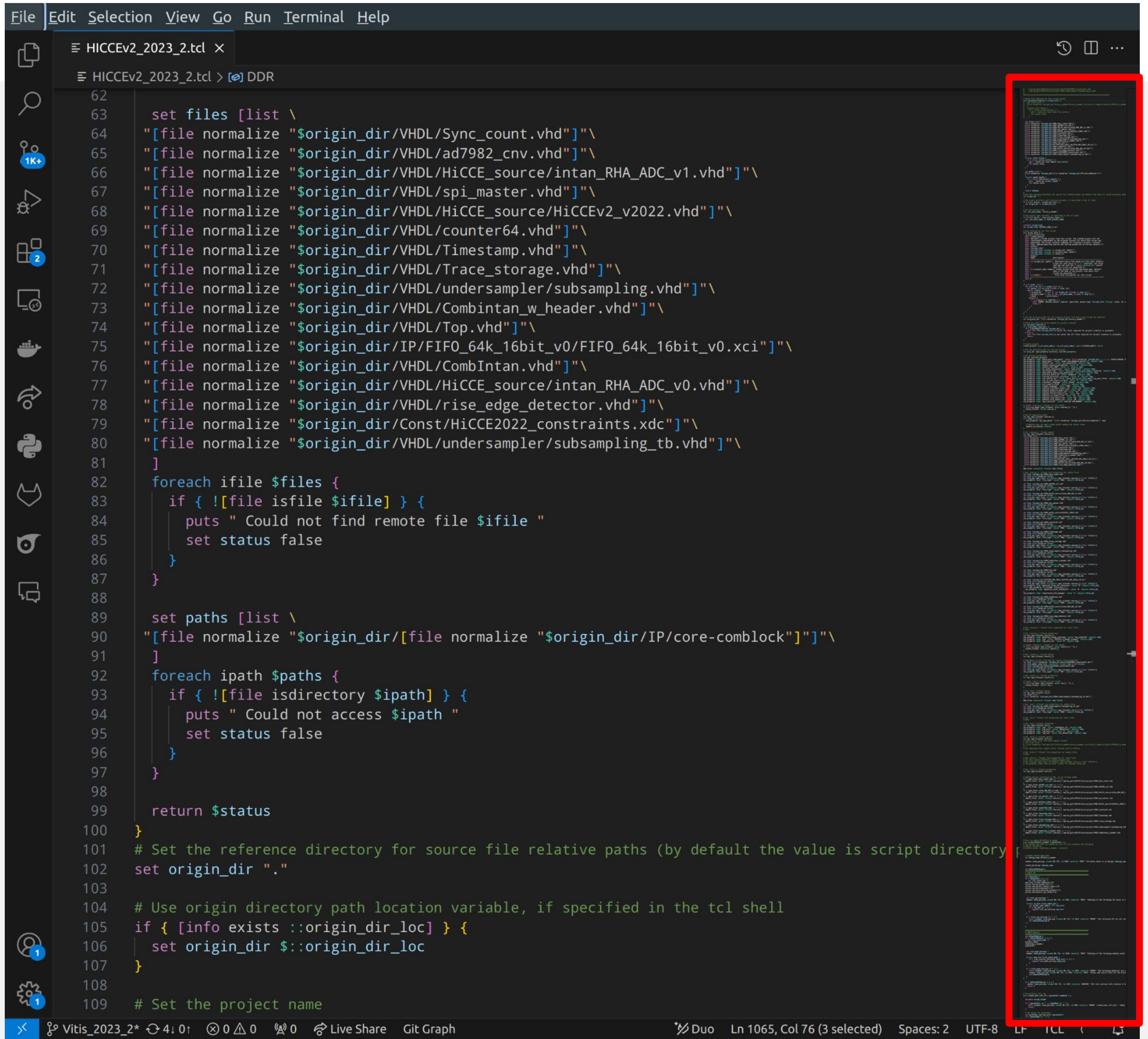




```
### Instantiating Comblock
create_bd_cell -type ip -vlnv www.ictp.it:user:comblock:2.0 MY_COMBLOCK
set_property -dict [list \
 CONFIG.REGS_IN_DEPTH {1} \
 CONFIG.REGS_OUT_DEPTH {2} \
 CONFIG.DRAM_IO_ENA {false} \
 CONFIG.FIFO_IN_ENA {false} \
] [get_bd_cells MY_COMBLOCK]
### Instantiating Flopoco division module
create_bd_cell -type ip -vlnv xilinx.com:ip:util_vector_logic:2.0 MY_AND_GATE
set_property CONFIG.C_SIZE {32} [get_bd_cells MY_AND_GATE]

# Block Diagarm Interconnection
# Comblock to Logic Vector
connect_bd_net [get_bd_pins MY_COMBLOCK/reg0_o] [get_bd_pins MY_AND_GATE/Op1]
connect_bd_net [get_bd_pins MY_COMBLOCK/reg1_o] [get_bd_pins MY_AND_GATE/Op2]
# Logic Vector to Comblock
connect_bd_net [get_bd_pins MY_AND_GATE/Res] [get_bd_pins MY_COMBLOCK/reg0_i]
```

The Frustrating Part

It can get very complicated very quickly.



```tcl

  set files [list \
 "[file normalize "$origin_dir/VHDL/Sync_count.vhd"]"\
 "[file normalize "$origin_dir/VHDL/ad7982_cnv.vhd"]"\
 "[file normalize "$origin_dir/VHDL/HiCCE_source/intan_RHA_ADC_v1.vhd"]"\
 "[file normalize "$origin_dir/VHDL/spi_master.vhd"]"\
 "[file normalize "$origin_dir/VHDL/HiCCE_source/HiCCEv2_v2022.vhd"]"\
 "[file normalize "$origin_dir/VHDL/counter64.vhd"]"\
 "[file normalize "$origin_dir/VHDL/Timestamp.vhd"]"\
 "[file normalize "$origin_dir/VHDL/Trace_storage.vhd"]"\
 "[file normalize "$origin_dir/VHDL/undersampler/subsampling.vhd"]"\
 "[file normalize "$origin_dir/VHDL/Combintan_w_header.vhd"]"\
 "[file normalize "$origin_dir/VHDL/Top.vhd"]"\
 "[file normalize "$origin_dir/IP/FIFO_64k_16bit_v0/FIFO_64k_16bit_v0.xci"]"\
 "[file normalize "$origin_dir/VHDL/CombIntan.vhd"]"\
 "[file normalize "$origin_dir/VHDL/HiCCE_source/intan_RHA_ADC_v0.vhd"]"\
 "[file normalize "$origin_dir/VHDL/rise_edge_detector.vhd"]"\
 "[file normalize "$origin_dir/Const/HiCCE2022_constraints.xdc"]"\
 "[file normalize "$origin_dir/VHDL/undersampler/subsampling_tb.vhd"]"\
  ]
  foreach ifile $files {
    if { ![file isfile $ifile] } {
      puts " Could not find remote file $ifile "
      set status false
    }
  }

  set paths [list \
 "[file normalize "$origin_dir/[file normalize "$origin_dir/IP/core-comblock"]"]"\
  ]
  foreach ipath $paths {
    if { ![file isdirectory $ipath] } {
      puts " Could not access $ipath "
      set status false
    }
  }

  return $status
}
# Set the reference directory for source file relative paths (by default the value is script directory
set origin_dir "."

# Use origin directory path location variable, if specified in the tcl shell
if { [info exists ::origin_dir_loc] } {
  set origin_dir $::origin_dir_loc
}

# Set the project name
```

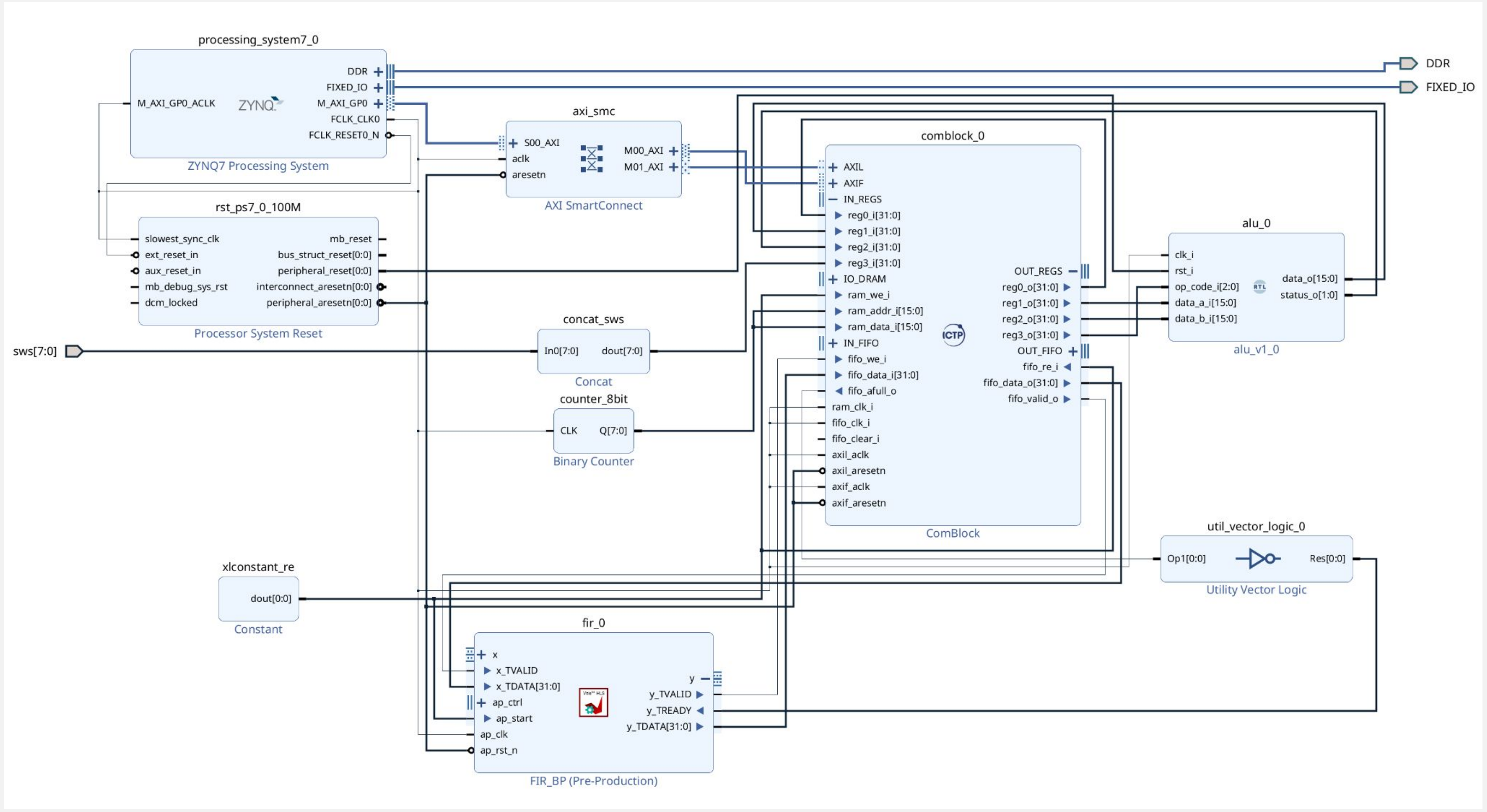
Vitis_2023_2* 4↓ 0↑ ⊗ 0 ⚠ 0 0 Live Share Git Graph
Duo Ln 1065, Col 76 (3 selected) Spaces: 2 UTF-8 LF TCL

You haven’t answered the question….

Why TCL?

Great for automation flow

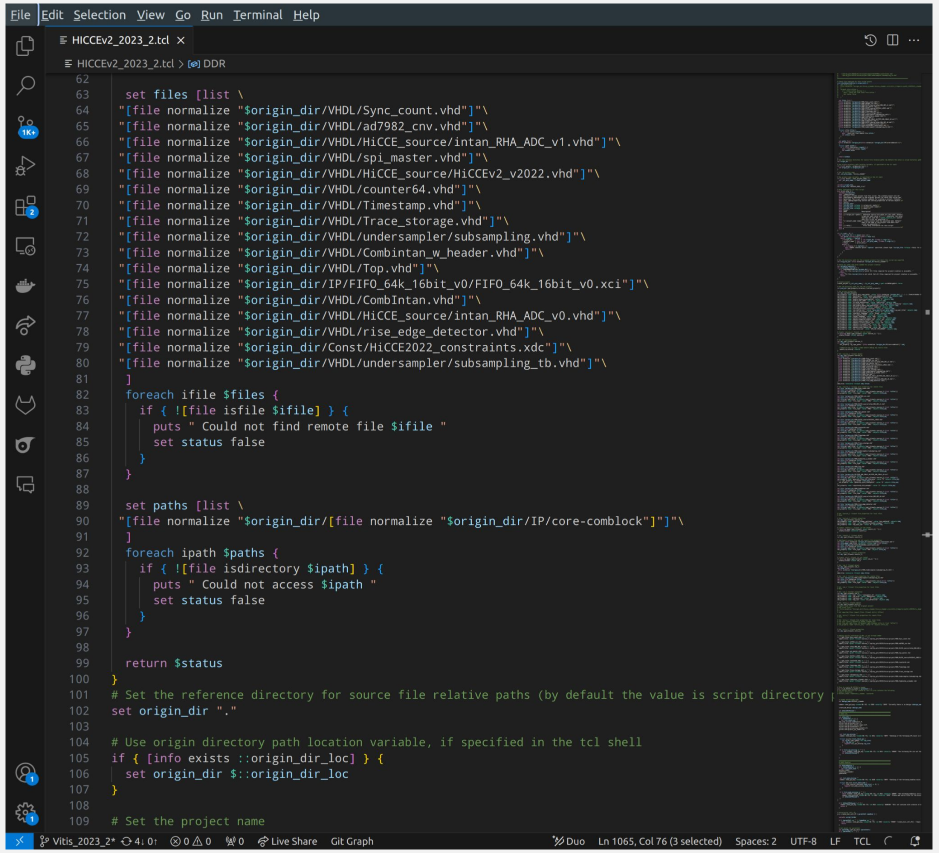
```
source ./my_project.tcl
```



You haven’t answered the question….

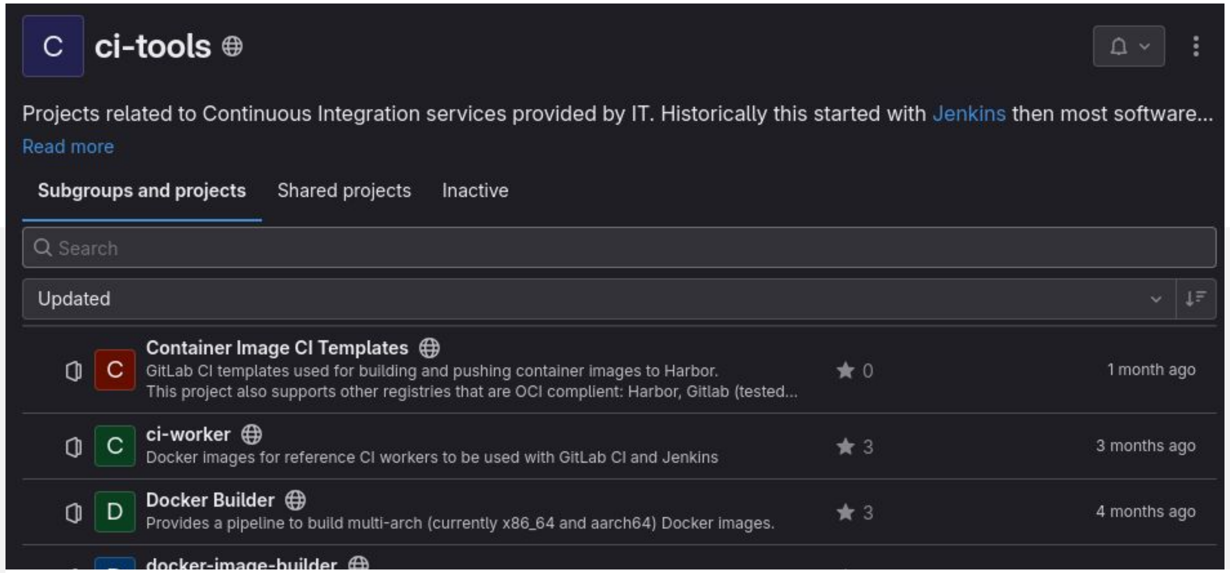
Why TCL?

Great for automation flow



```
  set files [list \
 "[file normalize "$origin_dir/VHDL/Sync_count.vhd"]"\
 "[file normalize "$origin_dir/VHDL/ad7982_cnv.vhd"]"\
 "[file normalize "$origin_dir/VHDL/HiCCE_source/intan_RHA_ADC_v1.vhd"]"\
 "[file normalize "$origin_dir/VHDL/spi_master.vhd"]"\
 "[file normalize "$origin_dir/VHDL/HiCCE_source/HiCCEv2_v2022.vhd"]"\
 "[file normalize "$origin_dir/VHDL/counter64.vhd"]"\
 "[file normalize "$origin_dir/VHDL/Timestamp.vhd"]"\
 "[file normalize "$origin_dir/VHDL/Trace_storage.vhd"]"\
 "[file normalize "$origin_dir/VHDL/undersampler/subsampling.vhd"]"\
 "[file normalize "$origin_dir/VHDL/Combintan_w_header.vhd"]"\
 "[file normalize "$origin_dir/VHDL/Top.vhd"]"\
 "[file normalize "$origin_dir/IP/FIFO_64k_16bit_v0/FIFO_64k_16bit_v0.xci"]"\
 "[file normalize "$origin_dir/VHDL/CombIntan.vhd"]"\
 "[file normalize "$origin_dir/VHDL/HiCCE_source/intan_RHA_ADC_v0.vhd"]"\
 "[file normalize "$origin_dir/VHDL/rise_edge_detector.vhd"]"\
 "[file normalize "$origin_dir/Const/HiCCE2022_constraints.xdc"]"\
 "[file normalize "$origin_dir/VHDL/undersampler/subsampling_tb.vhd"]"\
  ]
  foreach ifile $files {
    if { ![file isfile $ifile] } {
      puts " Could not find remote file $ifile "
      set status false
    }
  }

  set paths [list \
 "[file normalize "$origin_dir/[file normalize "$origin_dir/IP/core-comblock"]"]"\
  ]
  foreach ipath $paths {
    if { ![file isdirectory $ipath] } {
      puts " Could not access $ipath "
      set status false
    }
  }

  return $status
}
# Set the reference directory for source file relative paths (by default the value is script directory
set origin_dir "."

# Use origin directory path location variable, if specified in the tcl shell
if { [info exists ::origin_dir_loc] } {
  set origin_dir $::origin_dir_loc
}

# Set the project name
```

Vitis_2023_2* 4↓ 0↑ 0 0 0 Live Share Git Graph
Duo Ln 1065, Col 76 (3 selected) Spaces: 2 UTF-8 LF TCL



ci-tools

Projects related to Continuous Integration services provided by IT. Historically this started with Jenkins then most software...

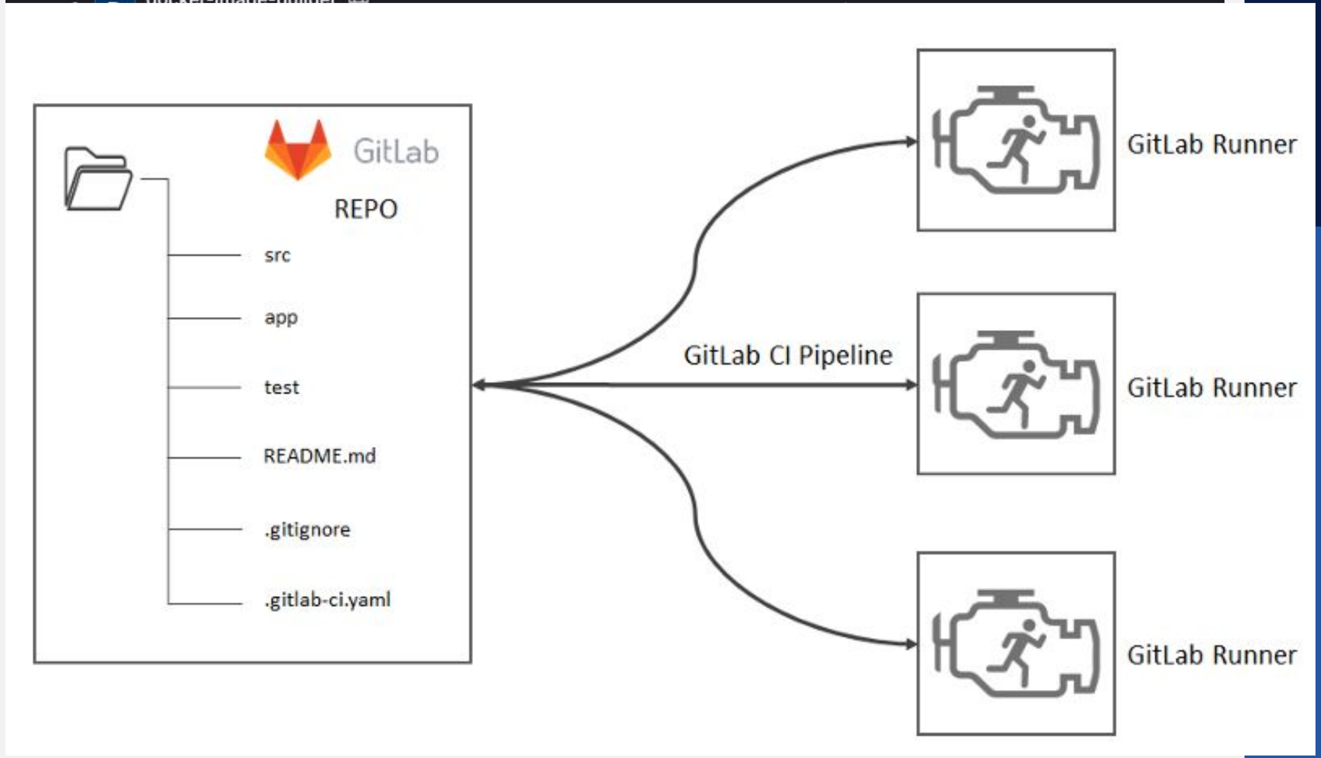
Read more

Subgroups and projects | Shared projects | Inactive

Search

Updated

Container Image CI Templates GitLab CI templates used for building and pushing container images to Harbor. This project also supports other registries that are OCI compliant: Harbor, Gitlab (tested...	★ 0	1 month ago
ci-worker Docker images for reference CI workers to be used with GitLab CI and Jenkins	★ 3	3 months ago
Docker Builder Provides a pipeline to build multi-arch (currently x86_64 and aarch64) Docker images.	★ 3	4 months ago
docker-image-builder		



The Abdus Salam International Centre for Theoretical Physics

Workshop on Fully Programmable Systems-on-Chip for Scientific Applications

We will continue in the lab.

Luis Guillermo García Ordóñez