RTML
Vivado HLS + LabVIEW

Author: Philip Conrad

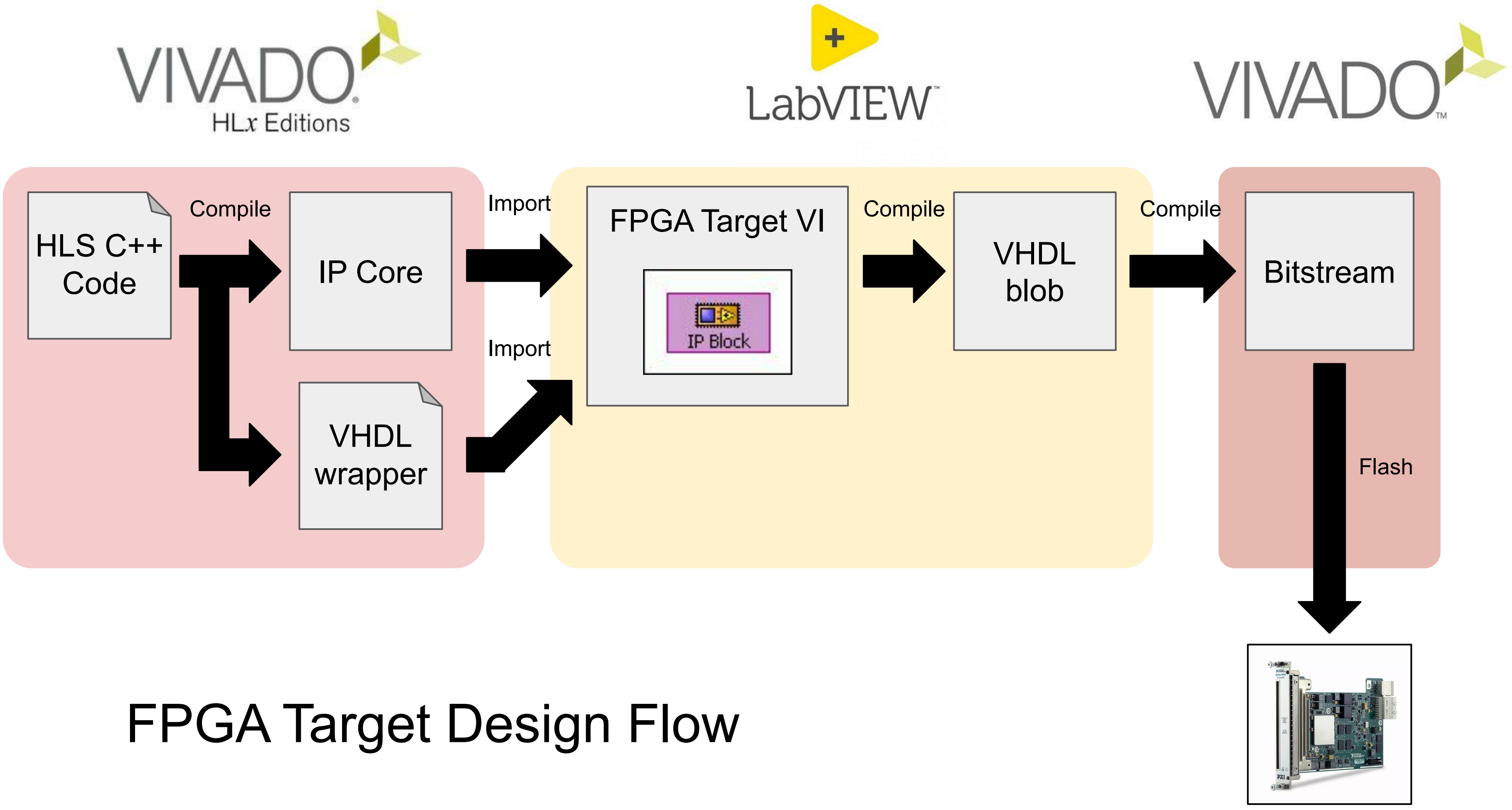
Overview

Overview of this guide

This guide covers:

- Vivado HLS compile/export flow, including wrapper script(s).
  - Covers: project setup, compilation, and export-to-rtl workflow.
  - Not Covered: testbench creation and usage
- Design of LabVIEW host/target VIs for streaming-style processing.
  - Makes use of DMA FIFOs.
  - Designer can drop the host-to-target DMA FIFO if only reading from FPGA.

Design Flow

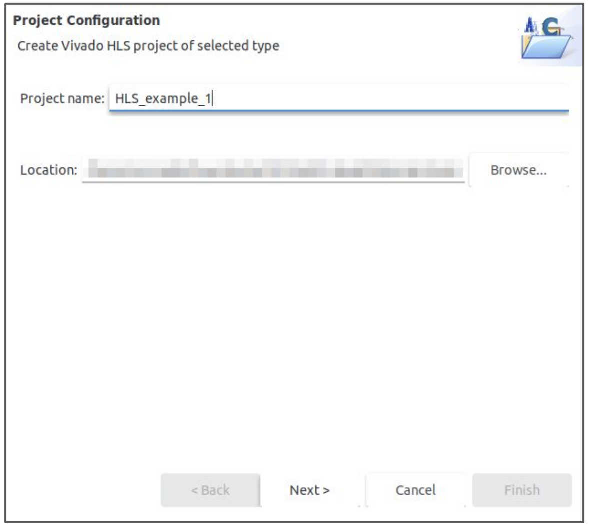


FPGA Target Design Flow

Vivado HLS Workflow

Getting Started - Creating a project (1/3)

- Open up Vivado HLS 2019.1.
- Create a new project with any name you like.



Project Configuration

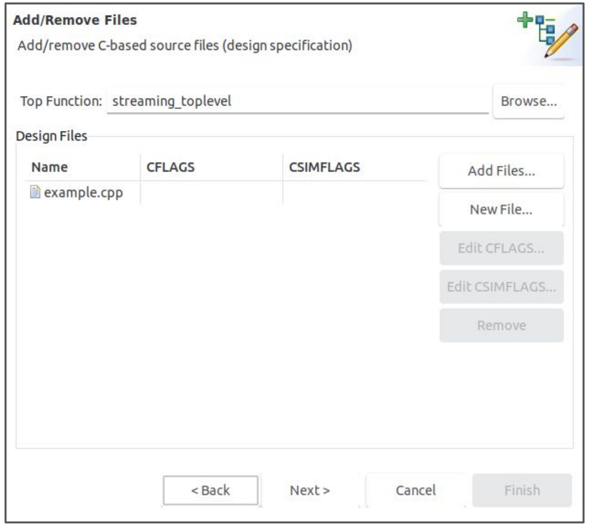
Create Vivado HLS project of selected type

Project name: HLS_example_1

Location: [redacted]

Getting Started - Creating a project (2/3)

- Add any relevant .c/.cpp files to the project.
- Add any testbench .c/.cpp files to the project on the next page (not shown here).



Add/Remove Files

Add/remove C-based source files (design specification)

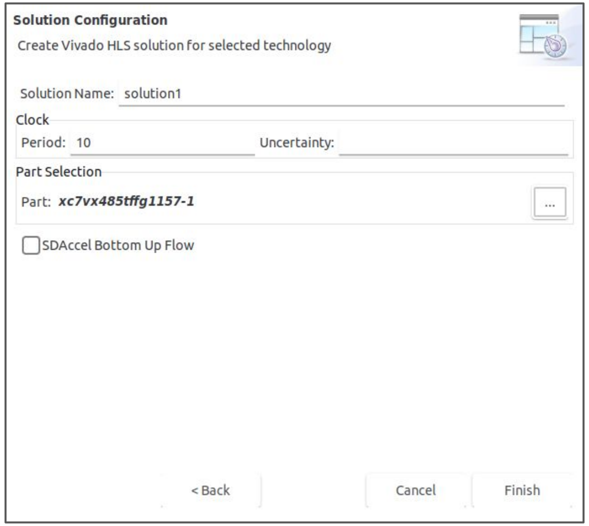
Top Function: streaming_toplevel

Design Files

Name	CFLAGS	CSIMFLAGS
example.cpp		

Getting Started - Creating a project (3/3)

- Where you see “Part Selection”, choose the FPGA chip appropriate for your work:
  Examples:
  - Kintex UltraScale 060 card → `xcku060-ffva1156-2-e`
  - Kintex-7 410T card → `xc7k410tfbv900-2`



Solution Configuration

Create Vivado HLS solution for selected technology

Solution Name: solution1

Clock

Period: 10 Uncertainty:

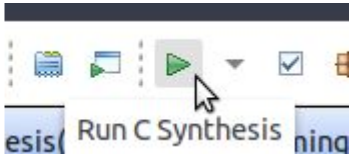
Part Selection

Part: ***xc7vx485tffg1157-1***

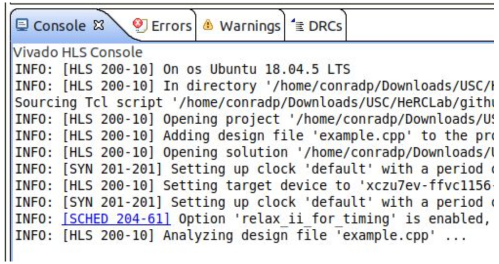
☐ SDAccel Bottom Up Flow

Getting Started - Compiling

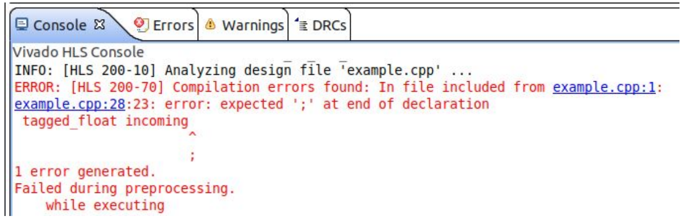
- Click the green synthesis button to start a compile job.



- Pay attention to the “Console” window.
    - If your C/C++ code has errors, they will appear in red in the Console Log.

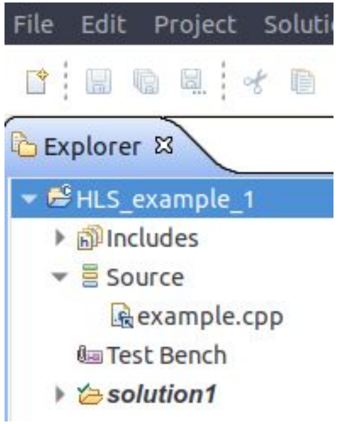


```
Vivado HLS Console
INFO: [HLS 200-10] On os Ubuntu 18.04.5 LTS
INFO: [HLS 200-10] In directory '/home/conradp/Downloads/USC/I
Sourcing Tcl script '/home/conradp/Downloads/USC/HeRCLab/githu
INFO: [HLS 200-10] Opening project '/home/conradp/Downloads/US
INFO: [HLS 200-10] Adding design file 'example.cpp' to the pro
INFO: [HLS 200-10] Opening solution '/home/conradp/Downloads/U
INFO: [SYN 201-201] Setting up clock 'default' with a period o
INFO: [HLS 200-10] Setting target device to 'xczu7ev-ffvc1156-
INFO: [SYN 201-201] Setting up clock 'default' with a period o
INFO: [SCHED 204-61] Option 'relax_ii_for_timing' is enabled,
INFO: [HLS 200-10] Analyzing design file 'example.cpp' ...
```



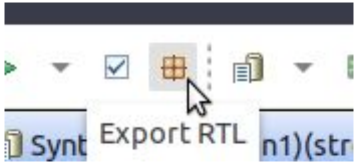
```
Vivado HLS Console
INFO: [HLS 200-10] Analyzing design file 'example.cpp' ...
ERROR: [HLS 200-70] Compilation errors found: In file included from example.cpp:1:
example.cpp:28:23: error: expected ';' at end of declaration
 tagged_float incoming
                      ^
                      ;
1 error generated.
Failed during preprocessing.
    while executing
```

- If you want to edit your code, it can be found under the “Source” category in the Project Explorer.

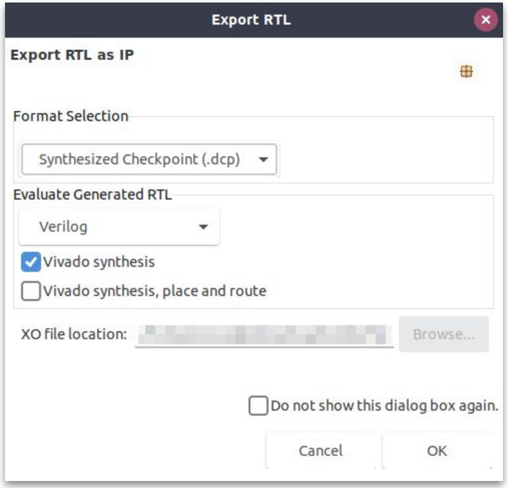


Getting Started - Export RTL (1/2)

- Click the little brown “Export RTL” button.



- Select the “Synthesized Checkpoint (.dcp)” option under the “Export RTL” menu.


Export RTL

Export RTL as IP

Format Selection

Synthesized Checkpoint (.dcp)

Evaluate Generated RTL

Verilog

[✓] Vivado synthesis

[ ] Vivado synthesis, place and route

XO file location:

Browse...

[ ] Do not show this dialog box again.

Cancel OK

- Wait until the export process finishes.

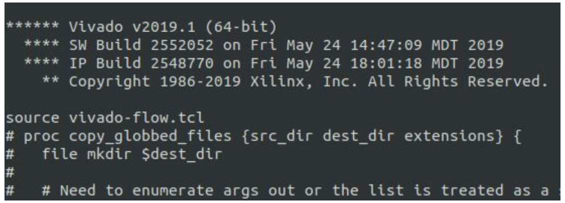
Getting Started - Export RTL (2/2)

You should see text like this printed out when running the script.

- In the `vivado-flow.tcl` script, look for:
  - `set PROJECT_NAME "HLS_example_1"`
  - `set IP_CORE_NAME "streaming_toplevel"`
  - Ensure these 2 lines match the name of your project/solution folder, and your top-level function name, respectively.
- Then, from a command line with Vivado tools available in your PATH, run the following command:

```
vivado -mode batch -source vivado-flow.tcl
```

- This should generate the final pair of files you need for LabVIEW under a folder named "`target`".

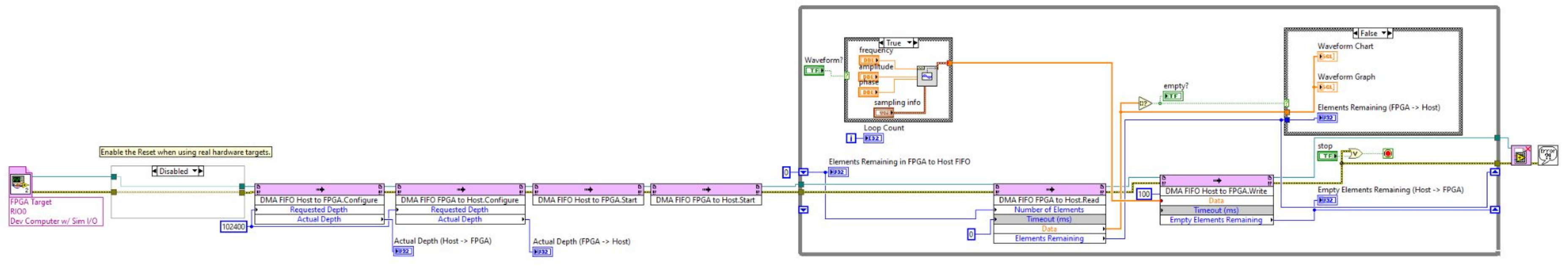


```
****** Vivado v2019.1 (64-bit)
  **** SW Build 2552052 on Fri May 24 14:47:09 MDT 2019
  **** IP Build 2548770 on Fri May 24 18:01:18 MDT 2019
    ** Copyright 1986-2019 Xilinx, Inc. All Rights Reserved.

source vivado-flow.tcl
# proc copy_globbed_files {src_dir dest_dir extensions} {
#   file mkdir $dest_dir
#
#   # Need to enumerate args out or the list is treated as a
```

LabVIEW VI Design - Host

Host-side VI (Overview)


Enable the Reset when using real hardware targets.
Disabled
FPGA Target
RIO0
Dev Computer w/ Sim I/O
102400
DMA FIFO Host to FPGA.Configure
Requested Depth
Actual Depth
DMA FIFO FPGA to Host.Configure
Requested Depth
Actual Depth
Actual Depth (Host -> FPGA)
Actual Depth (FPGA -> Host)
DMA FIFO Host to FPGA.Start
DMA FIFO FPGA to Host.Start
Waveform?
True
frequency
amplitude
phase
sampling info
Loop Count
Elements Remaining in FPGA to Host FIFO
0
0
DMA FIFO FPGA to Host.Read
Number of Elements
Timeout (ms)
Data
Elements Remaining
empty?
100
DMA FIFO Host to FPGA.Write
Data
Timeout (ms)
Empty Elements Remaining
False
Waveform Chart
Waveform Graph
Elements Remaining (FPGA -> Host)
stop
Empty Elements Remaining (Host -> FPGA)

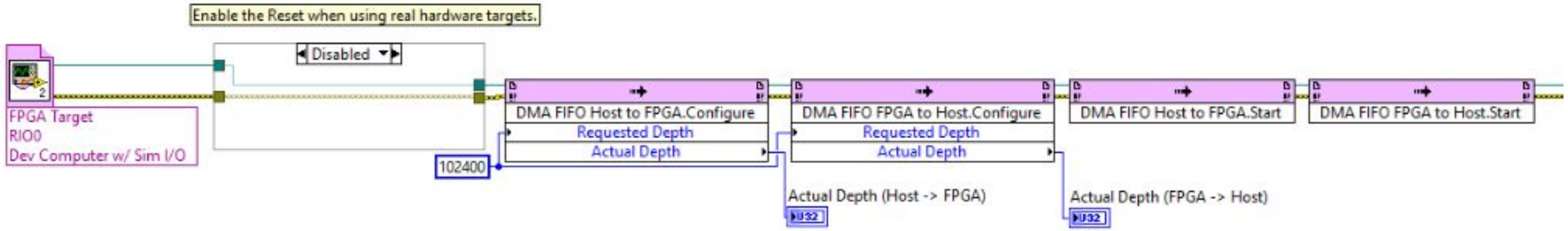
Setup

Main DMA Loop

Host-side VI (Setup)

Enable/Disable block for Reset

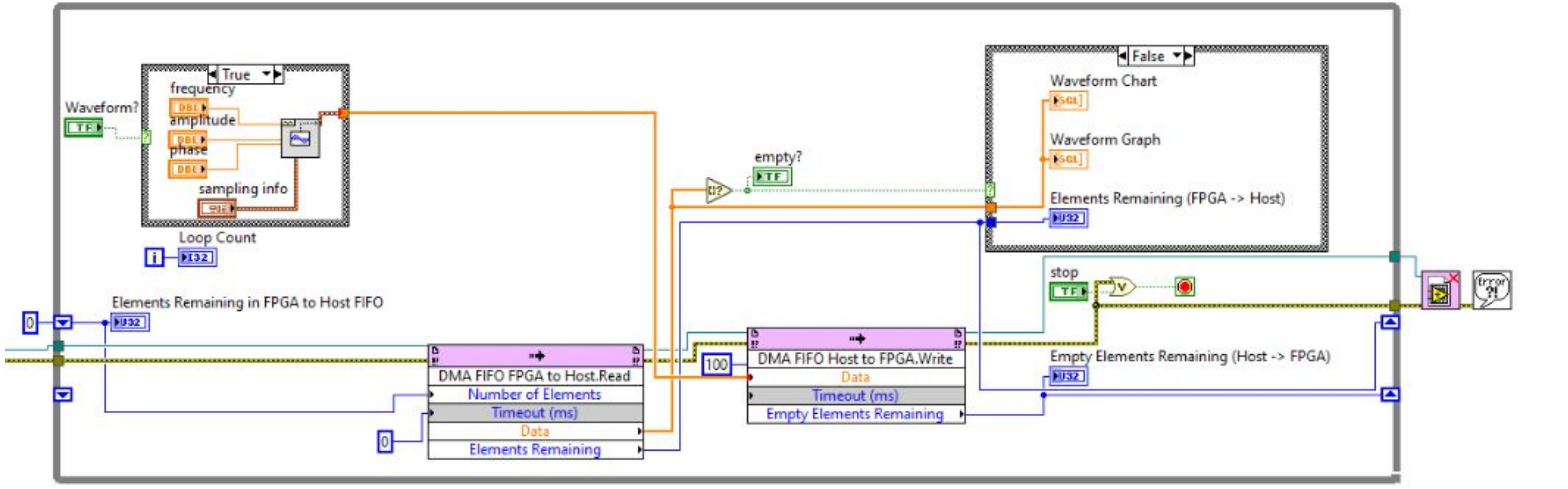
Init Host-side FIFOs (warm start)


Enable the Reset when using real hardware targets.
Disabled
FPGA Target
RIO0
Dev Computer w/ Sim I/O
102400
DMA FIFO Host to FPGA.Configure
Requested Depth
Actual Depth
DMA FIFO FPGA to Host.Configure
Requested Depth
Actual Depth
DMA FIFO Host to FPGA.Start
DMA FIFO FPGA to Host.Start
Actual Depth (Host -> FPGA)
U32
Actual Depth (FPGA -> Host)
U32

Open FPGA VI

Host-side FIFO Buffer Allocs

Host-side VI (Main DMA loop)



Waveform Generation

DMA Read

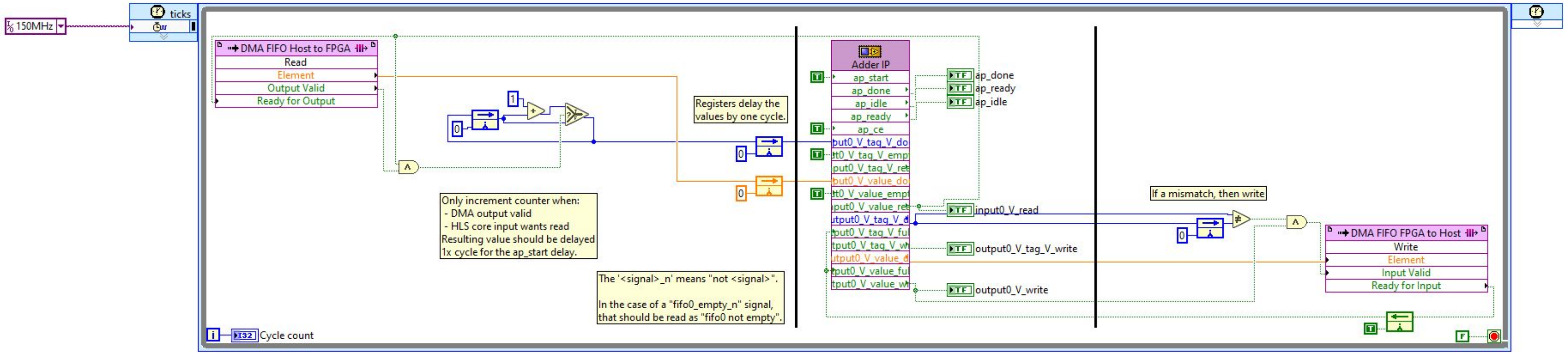
DMA Write

Front Panel Display

Close FPGA VI

LabVIEW VI Design - FPGA Target

FPGA Target-side VI (Overview)



Registers delay the values by one cycle.

Only increment counter when:
- DMA output valid
- HLS core input wants read

Resulting value should be delayed 1x cycle for the ap_start delay.

The '<signal>_n' means "not <signal>".

In the case of a "fifo0_empty_n" signal, that should be read as "fifo0 not empty".

If a mismatch, then write

DMA Read

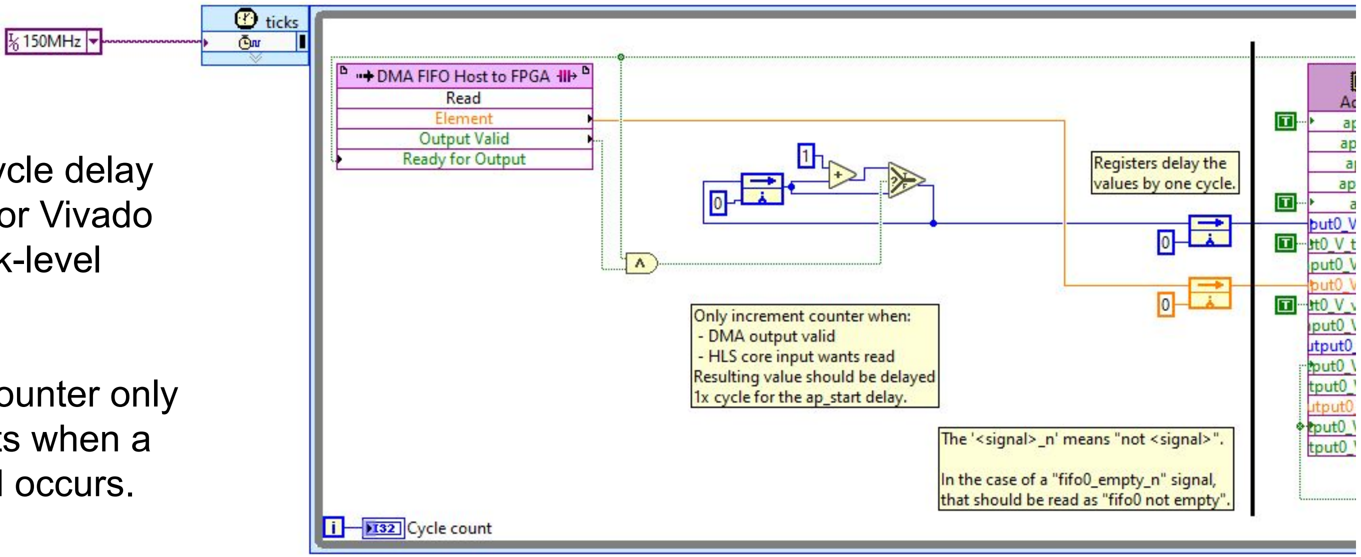
HLS IP Core

DMA Write

FPGA Target-side VI (DMA Read)

Initial 1-cycle delay required for Vivado HLS Block-level protocol.

The tag counter only increments when a DMA read occurs.



Registers delay the values by one cycle.

Only increment counter when:
- DMA output valid
- HLS core input wants read
Resulting value should be delayed 1x cycle for the ap_start delay.

The '<signal>_n' means "not <signal>".

In the case of a "fifo0_empty_n" signal, that should be read as "fifo0 not empty".

DMA Read (Handshaking)

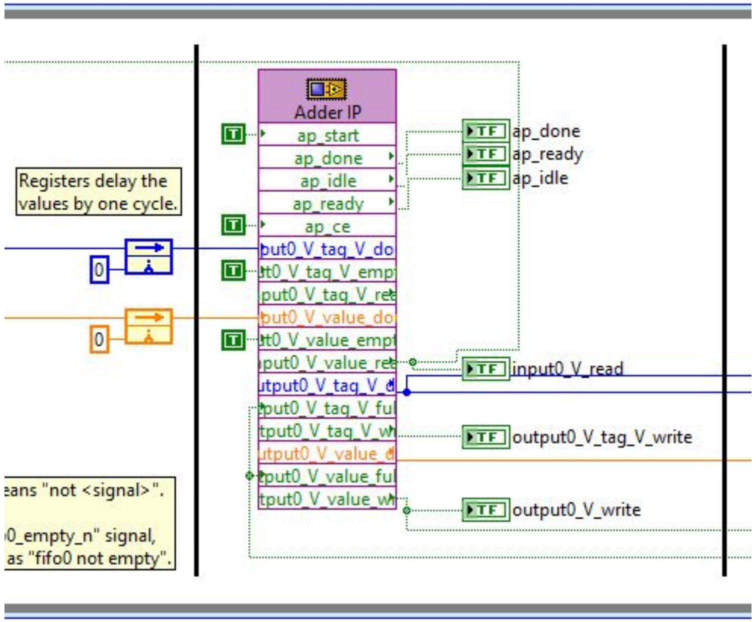
Counter Generator for the Tag values

1-cycle Delay Registers

FPGA Target-side VI (HLS Core)

The control signals are all asserted to `True` on the IP block.

The IP block is free-running, and counter values (tags) determine valid/invalid state of result values.

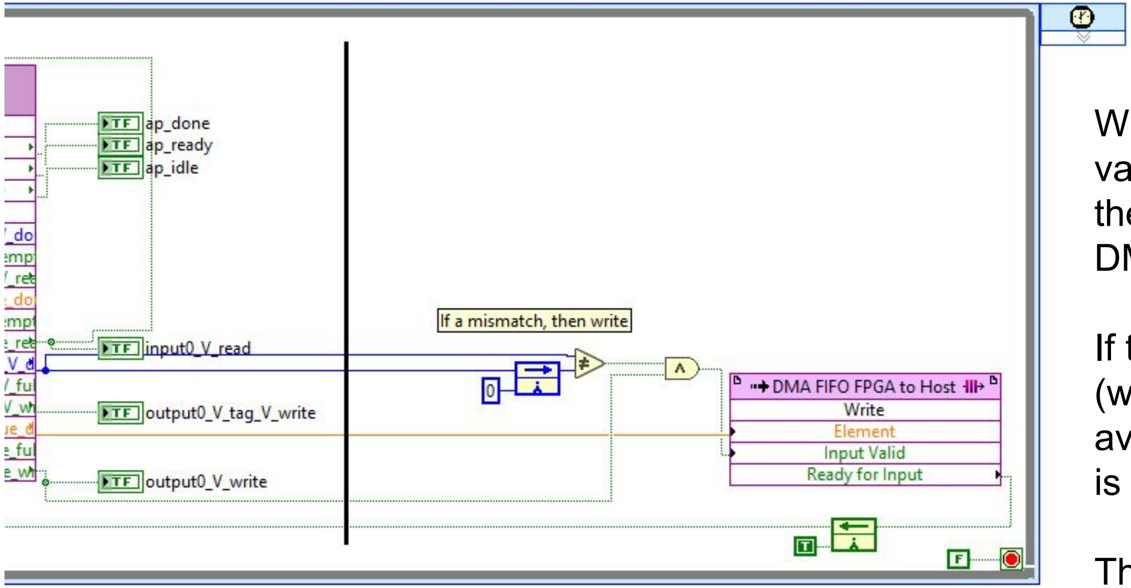


Delay registers

HLS IPI Node (IPIN)

Output wires

FPGA Target-side VI (DMA Write)



Output wires

Output Demangler

DMA Write (Handshaking)

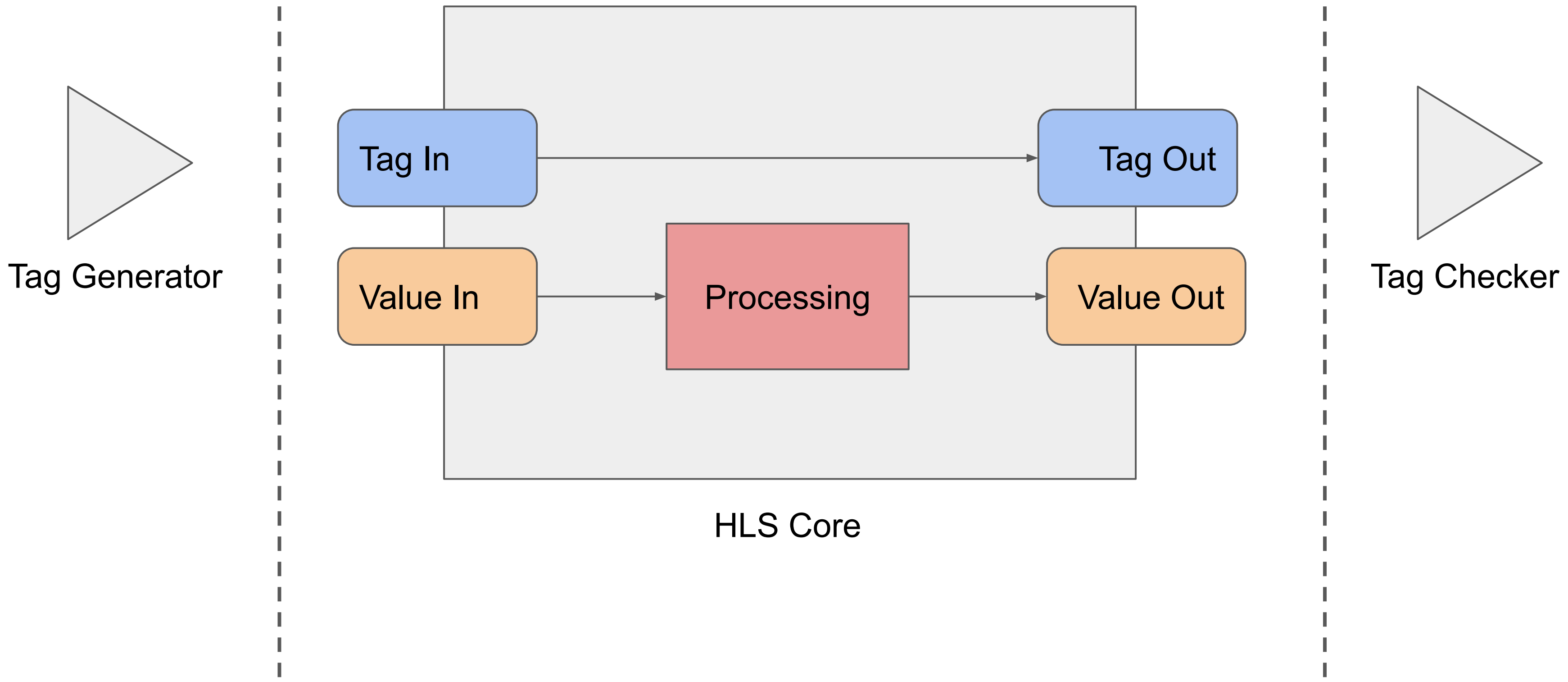
When the tag output value is different from the previous cycle, DMA write is enabled.

If the tag freezes (when no data available), DMA write is disabled.

This is a hack, but it works.

Extra (Unfinished) Slides

HLS Programming Model

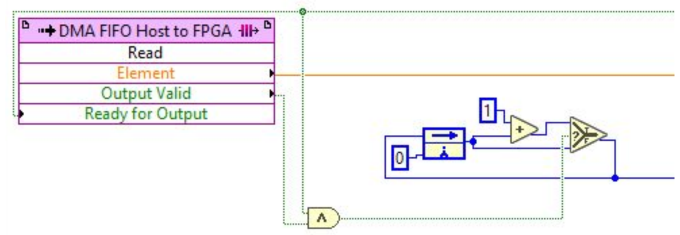


Tag System

Generator

- Increments the tag counter each time a valid sample is read into the HLS core.

- Disabled when HLS input FIFO disabled.
- Disabled when DMA input FIFO disabled.

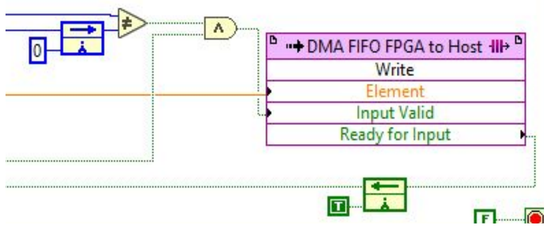


Checker

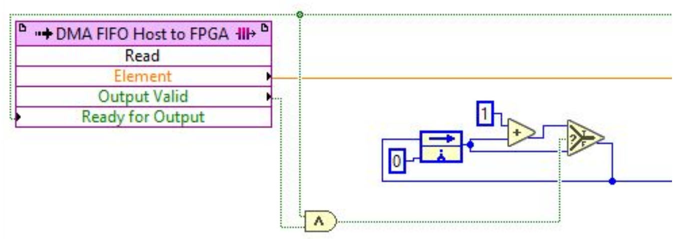
- Keeps last-seen tag value in a register.
- Compares last-seen to current tag value.

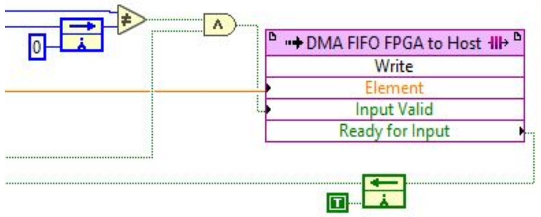
- On match, reject data value.
- On mismatch, data value must be new, so send to DMA output FIFO.

- Disabled when HLS output FIFO disabled.
- Disabled when DMA output FIFO disabled.



Tag System





- The FIFO DMA system uses normal LabVIEW flow-control.
- To play nice with LabVIEW, we have to jerry-rig a flow-control system ourselves.
- We use a tag system to mark data as interesting/not-interesting
    - Ensures the output DMA FIFO doesn’t get flooded with bogus values.
    - Has no effect on the input DMA FIFO.