High-Level FPGA Design

Lab 5: RedPitaya Project with Gain IP

In this lab you will:

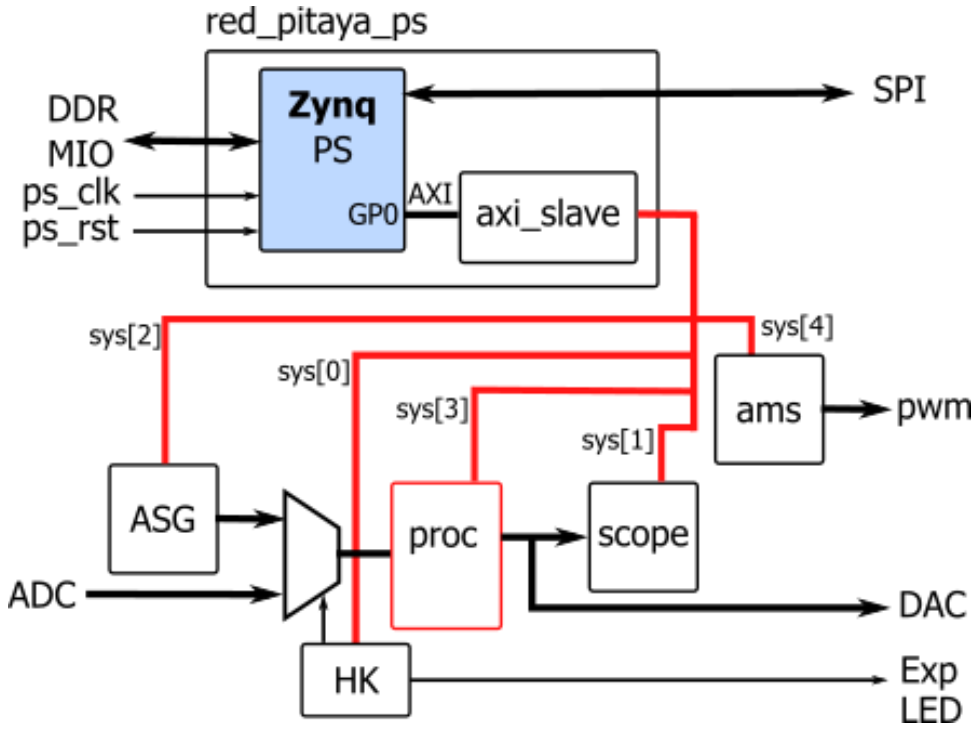
- Use modified FPGA project for RedPitaya
- Include gain IP to the Vivado project
- Compile Vivado project and prepare binary bitstream
- Upload bitstream to RedPitaya and test gain IP operation

5-1 Red Pitaya Project

- Open provided Vivado Red Pitaya Project

```
vivado/
 ├── rp25/    ← Vivado RedPitaya project
```

- Check project structure



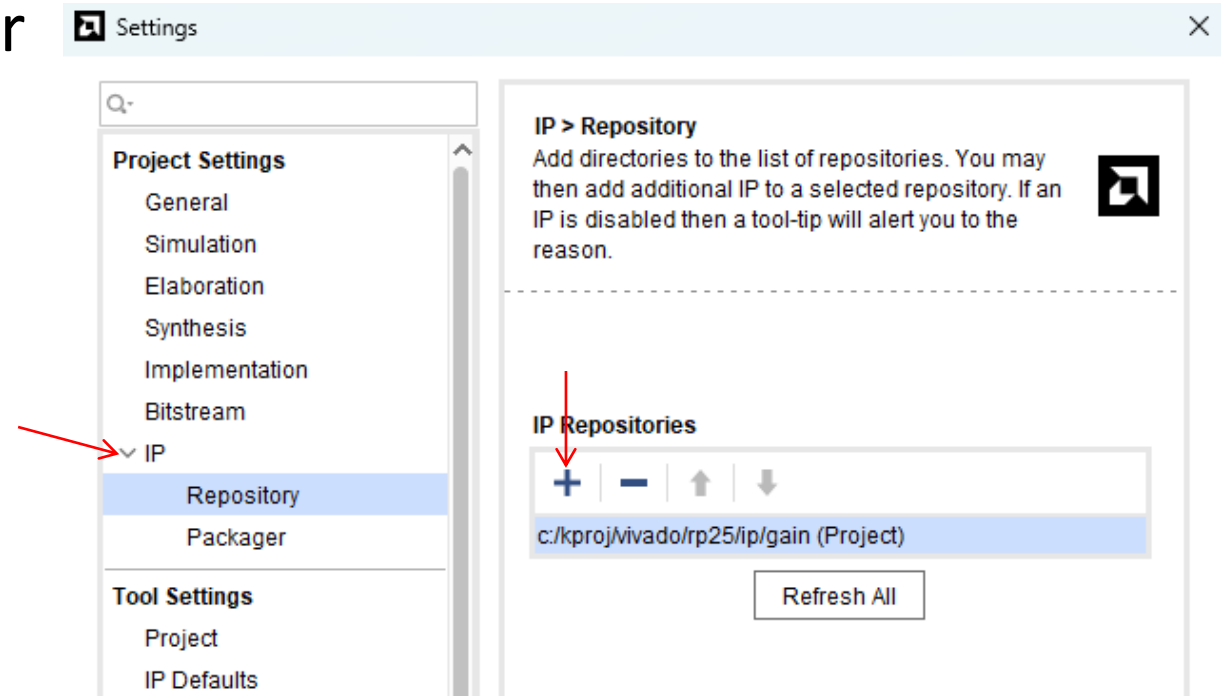
5-2 Set IP Repository with HLS Gain IP

- Locate HLS Gain IP component folder

  `proj\vitis\gain_ip`

- unzip **gain_ip.zip**

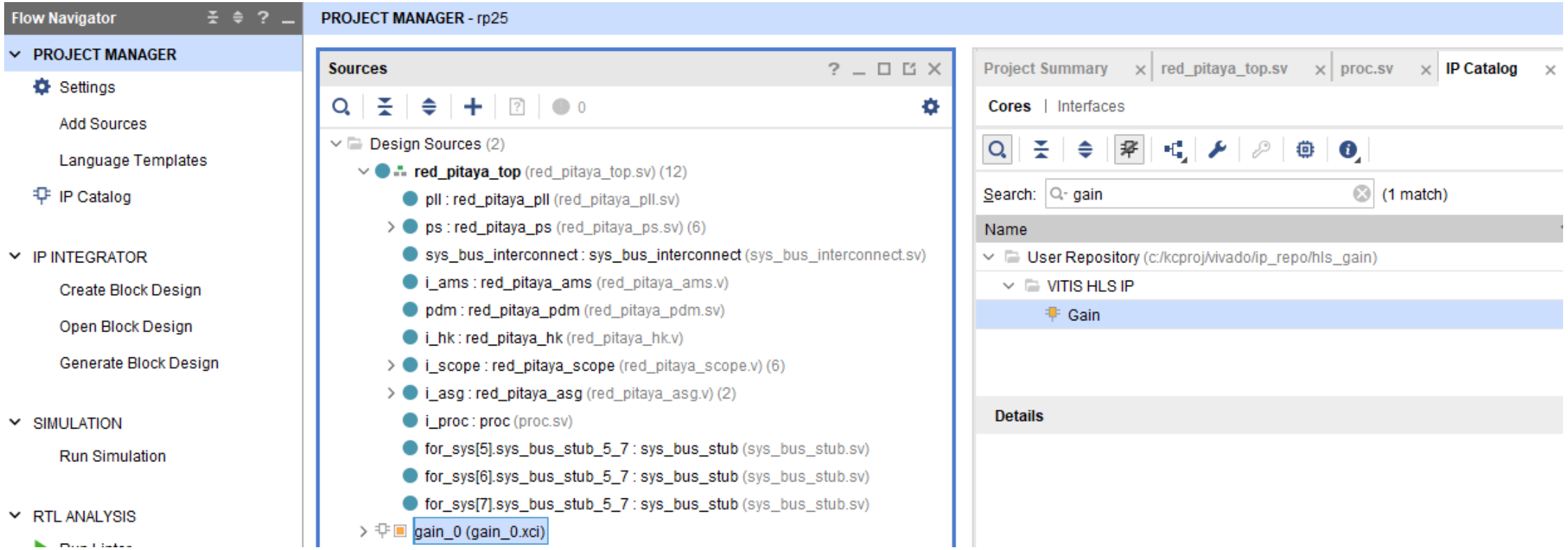
```
vivado/
 ├─ rp25/          ← project
    └─ip/          ← IPs
       └─gain/
          └─component.xml
```



- **Project Settings,** IP, Repository +
- Browse and add your hls_gain IP
  - Vivado reports adding 1 repository

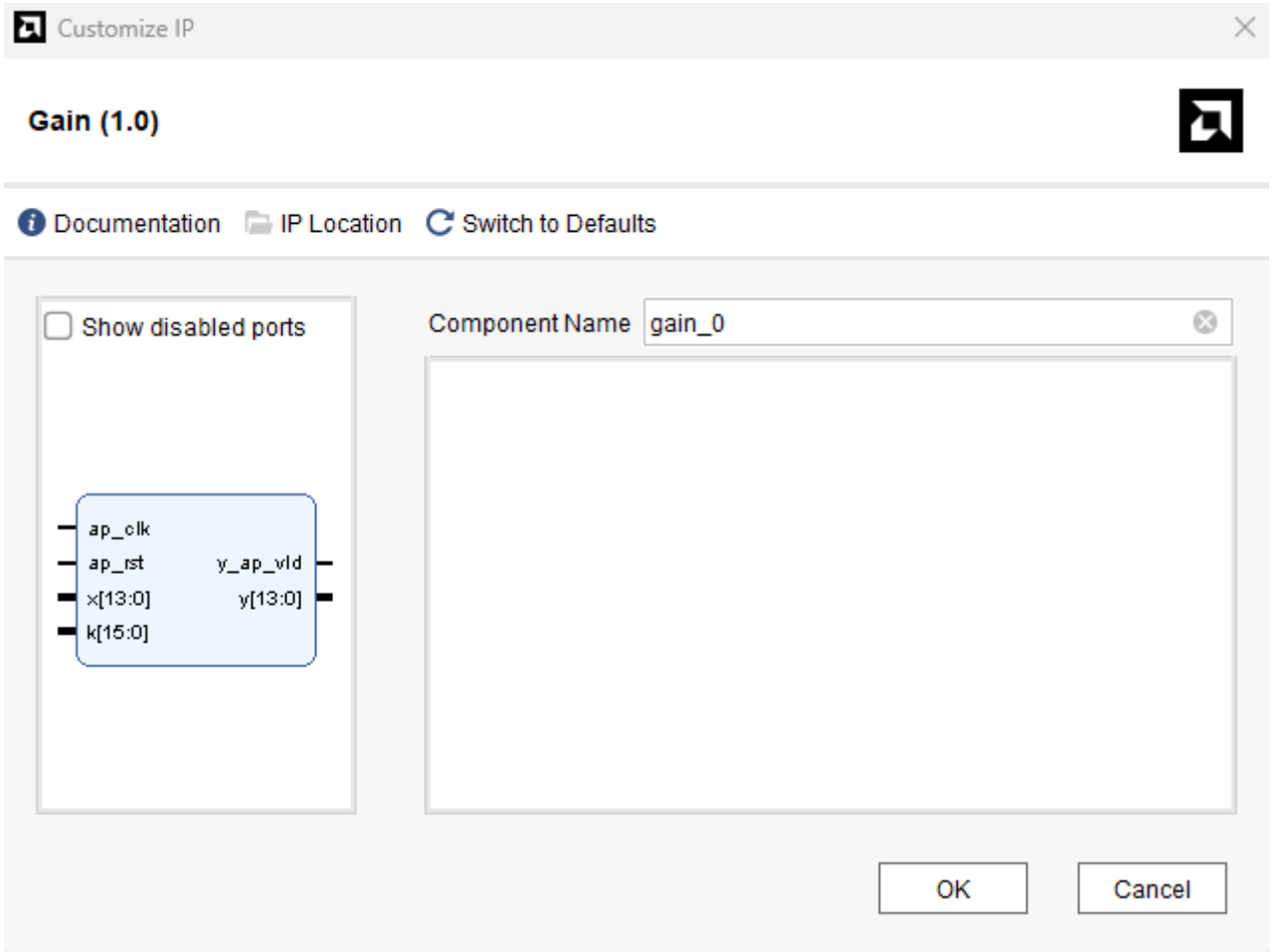
5-2 Add Gain IP to the project

- Click on IP Catalog in Flow Navigator and write Gain in Search window
- Double click Gain to include IP to the project, confirm and click Generate



5-2 Gain IP Module

- Check component name **gain_0** and ports

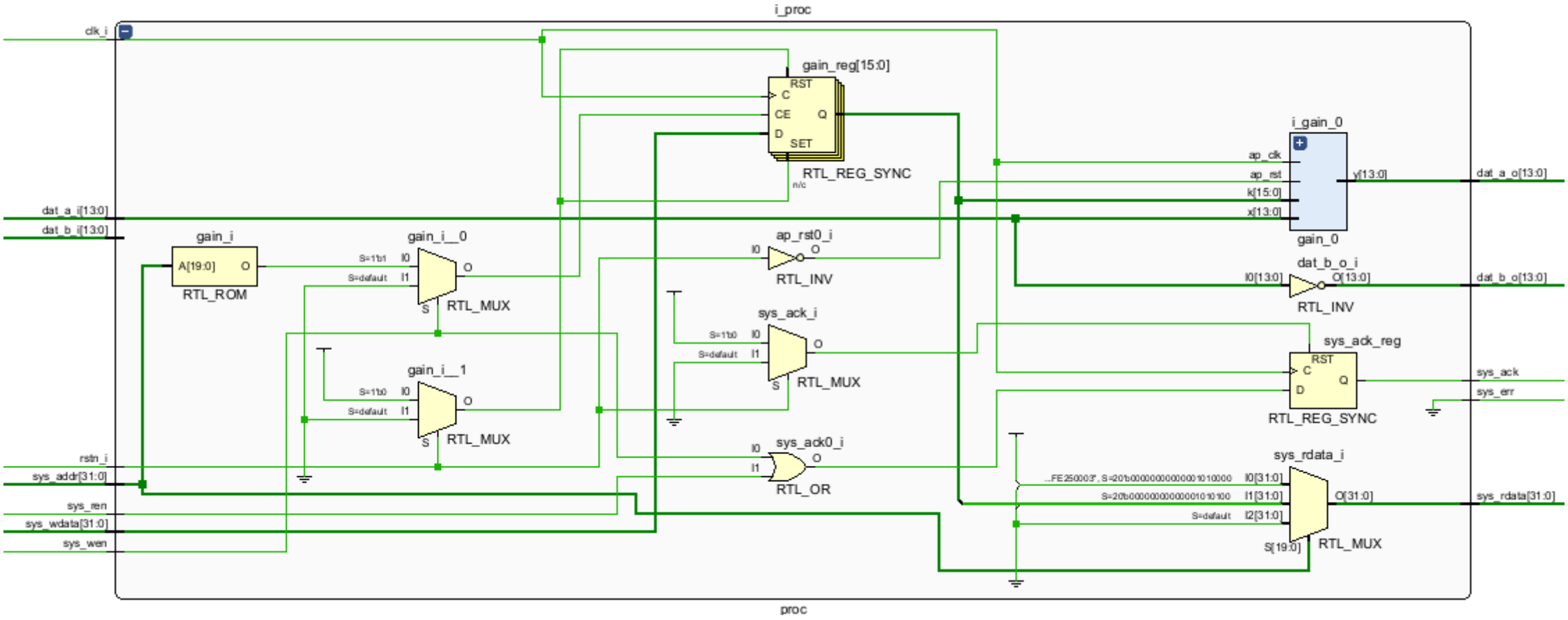


5-3 Module proc.sv

- two inputs connected to function generator (ASG) or ADC
- two outputs connected to scope and DAC
- There is additional logic to decode system bus. The module proc is connected to sys[3] which defines upper 12 address bits to be 0x403 and the lower bits are decoded in the module description:
  - 0x40300050 contains read-only register with fixed value: 0xFE250003
  - 0x40300054 contains 16-bit gain register

5-3 Modify proc.sv

- Connect gain_0
  - x => dat_a_i, k => gain, y => dat_a_o

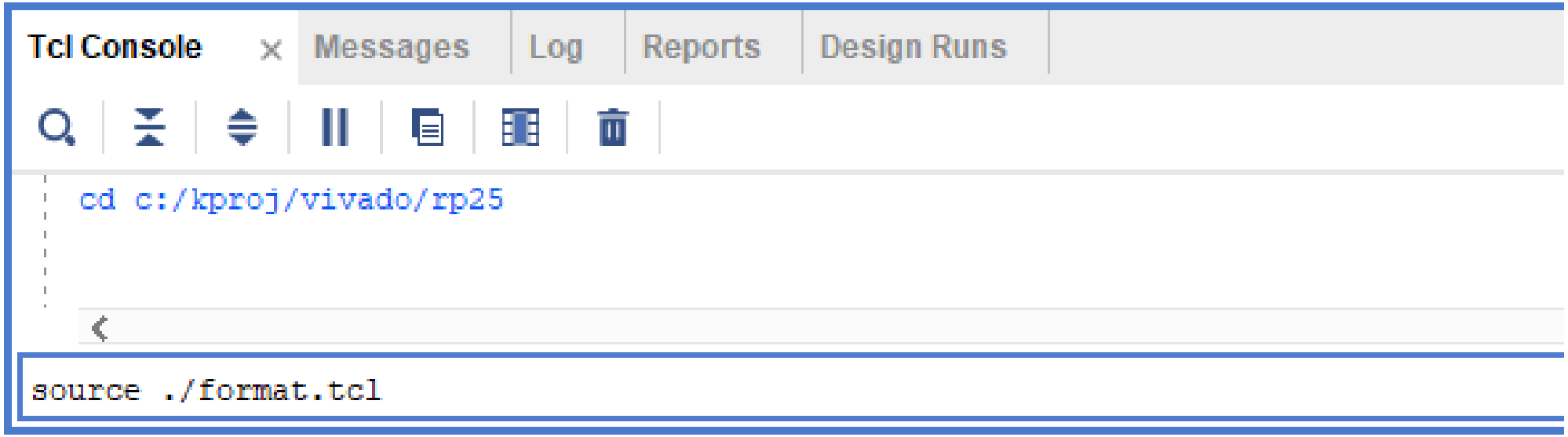


5-4 Synthesize and generate bitstream

- Generate Bitstream
  - Runs Synthesis
  - Implementation and
  - Generate Bitstream

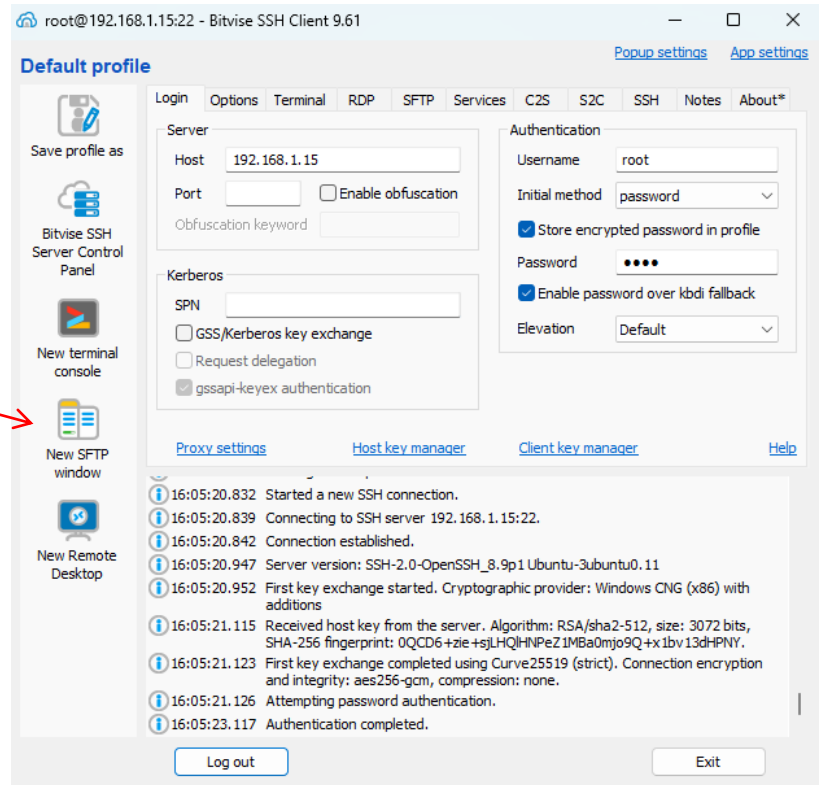
Note: the process takes 5-10 minutes!

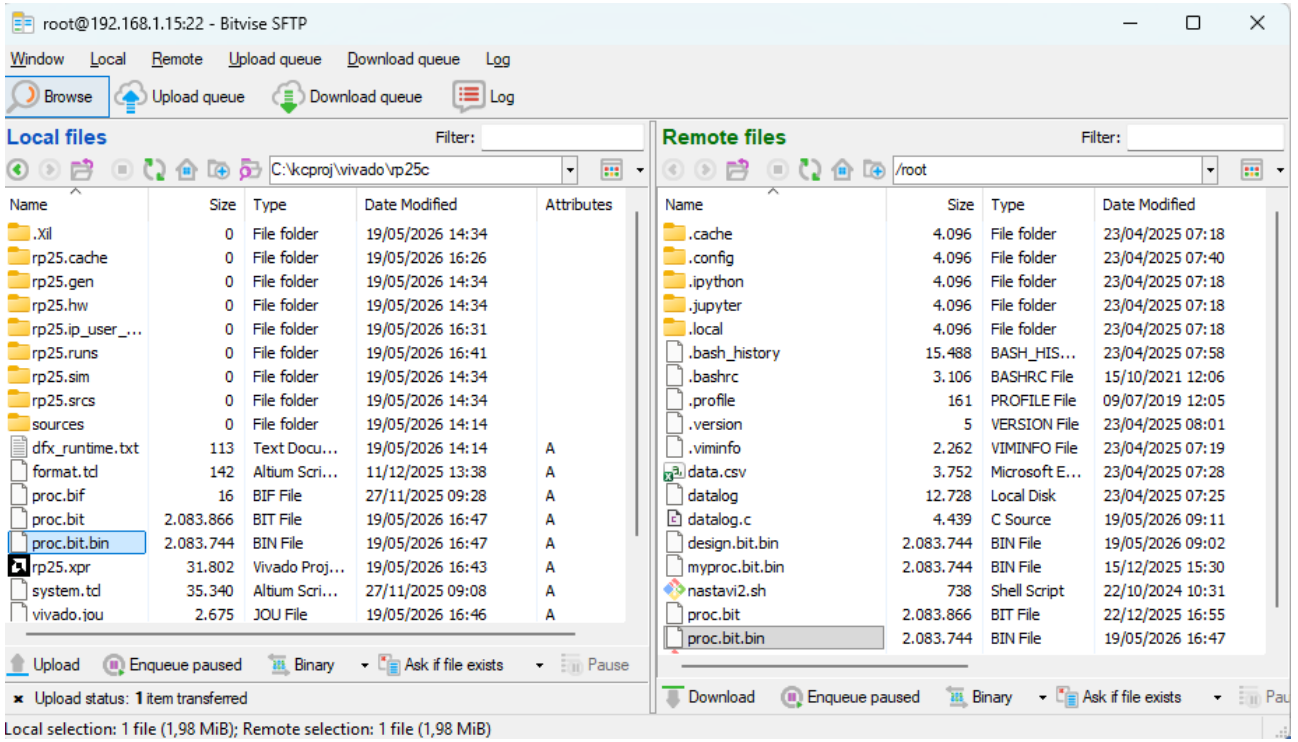
- In TCL Console change directory to your project folder and run tcl script



5-5 Connect Red Pitaya and Upload Bitstream

- Use Bitwise SSH Client to connect to RedPitaya boad
  - use local network, Ethernet adapter set to 192.168.1.1
  - Login HOST: 192.168.1.15, default username and pass: root
- Open New SFTP window and upload **proc.bit.bin**

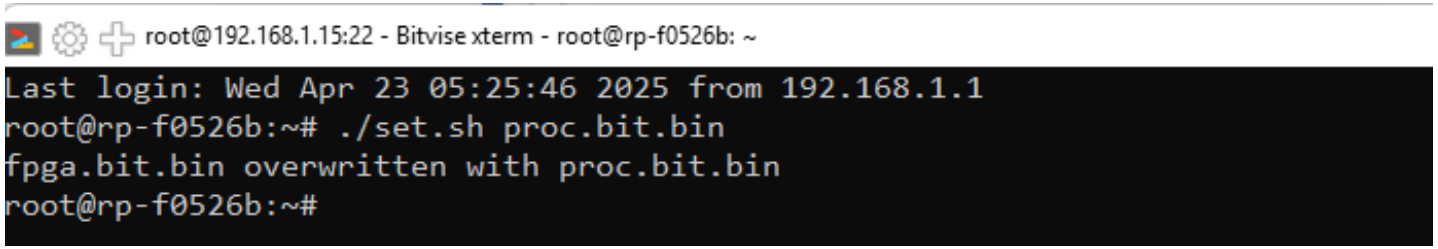




5-5 Set Bitstream

- In Bitwise SSH Client Open New terminal console and run:

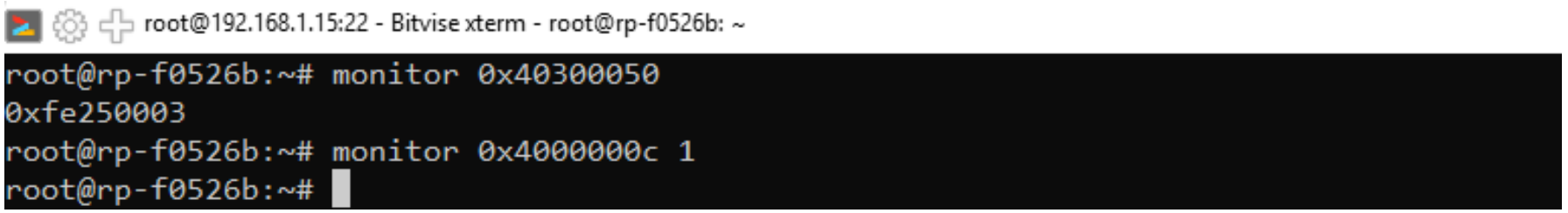
  ./set.sh proc.bit.bin



- set.sh overwrites default FPGA bitstream with your uploaded bitstream
- FPGA configuration will be executed by opening web application

5-6 Test with Web Application

- Open web browser at http://192.168.1.15/
- Click Oscilloscope Application
- Check in **terminal** console that your FPGA bitstream is loaded. Monitor content of register at address 0x40300000, the response should be 0xfe250003



5-6 Generate and observe signals

- Using **terminal** set data_loop: monitor 0x4000000C 1
- Generate sine wave: 1 kHz, 0.5 V at OUT1
- Observe IN1, OUT1 and IN2 (vertical setting 500 mV)

5-6 Generate and observe signals

- change gain setting: monitor 0x40300054 500
  - observe outputs with different gain settings
  - observe saturation (e.g. gain = 2500)

High-Level FPGA Design

Lab 6: RedPitaya Project with Averaging filter

In this lab you will:

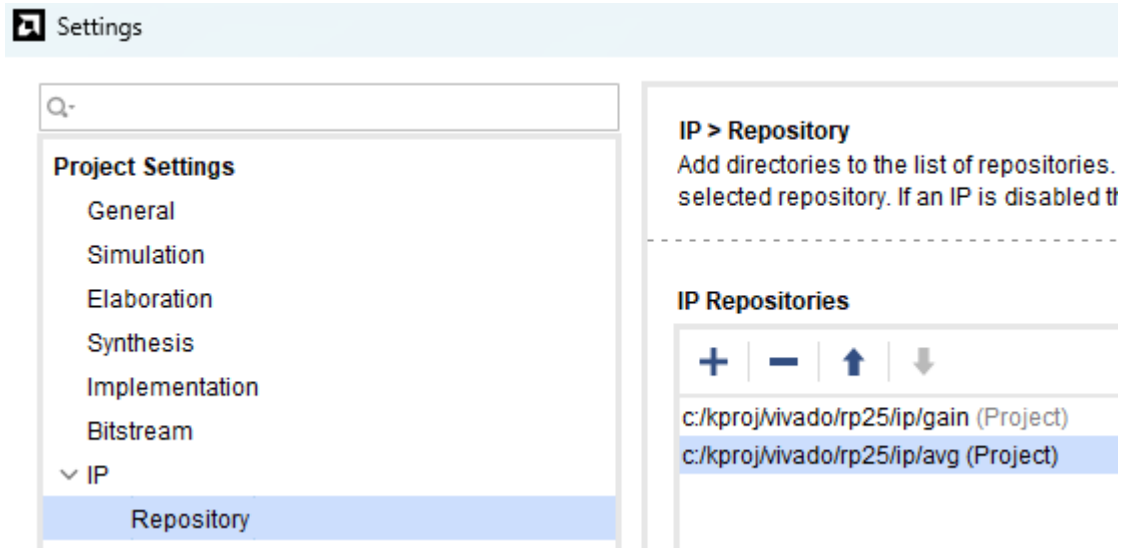
- Use modified FPGA project for RedPitaya
- Include average filter IP to the Vivado project
- Compile Vivado project and prepare binary bitstream
- Upload bitstream to RedPitaya and test filter operation

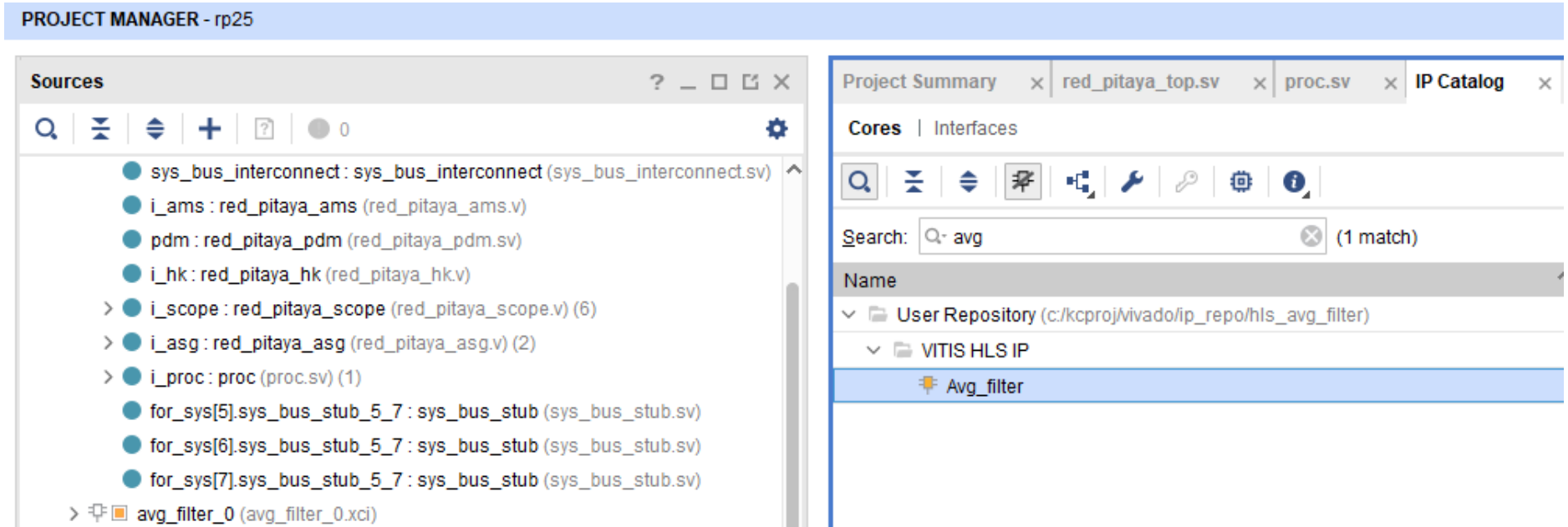
6-1 Prepare IP Component

- Open avg_filter component in Vitis
- Run C Synthesis
- Run PACKAGE
  - set output file: avg_ip
- unzip **avg_ip.zip** from `proj\vitis\avg_filter`
  - extract to folder `vivado\rp25\ip\avg`

6-1 Add Average Filter IP to Red Pitaya Project

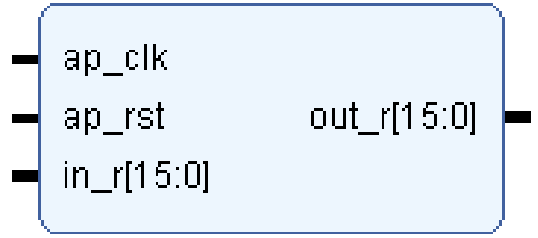
- Open Red Pitaya Vivado project
- Settings, Add IP Repository
- Open IP Catalog
- Add **Avg_filter** IP to the Project





6-2 Modify proc.sv

- Instantiate avg_filter_0 in proc.sv




```
logic signed [13:0] filt_i, filt_o;

assign filt_i = gain_o;

avg_filter_0 i_filt (
   .ap_clk    (clk_i),
   .ap_rst    (~rstn_i),
   .in_r      (filt_i),
   .out_r     (filt_o)
 );

assign dat_b_o = filt_o;  // filtered data to CH2
```

6-2 Modify proc.sv

- Modify ID in proc.sv
  - this is useful for tracking bitstream

```
always_comb begin
        unique case (sys_addr[19:0])
            20'h00050: sys_rdata = 32'hFE250005;      // read ID
            20'h00054: sys_rdata = {16'h0000, gain}; // read gain register
            default:   sys_rdata = 32'h00000000;
        endcase
    end
```

6-3 Synthesize and Generate Bitstream

- Generate Bitstream
  - Runs Synthesis, Implementation and Generate Bitstream
- In TCL Console change directory to your project folder and run tcl script `source ./format.tcl` producing final **proc.bit.bin**

6-4 Upload to Red Pitaya

- Connect to Red Pitaya with Bitwise SSH Client
- upload **proc.bit.bin**
- open New terminal console

  ./set.sh proc.bit.bin

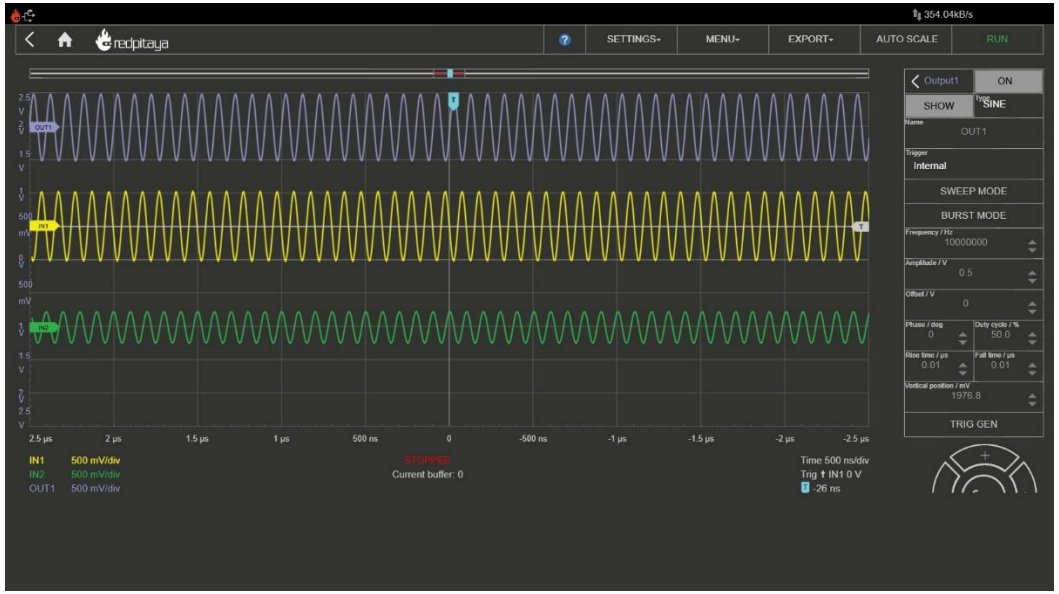
  Note: to actually change FPGA content, you should close Oscilloscope web application (if it is open) and open it again!

6-5 Test with Web Application

- Open web browser at http://192.168.1.15/
- Click Oscilloscope Application
- Check in **terminal** console that your FPGA bitstream is loaded. Monitor content of register at address 0x40300050, the response should be 0xfe250005
- Using **terminal** set data_loop: monitor 0x4000000C 1

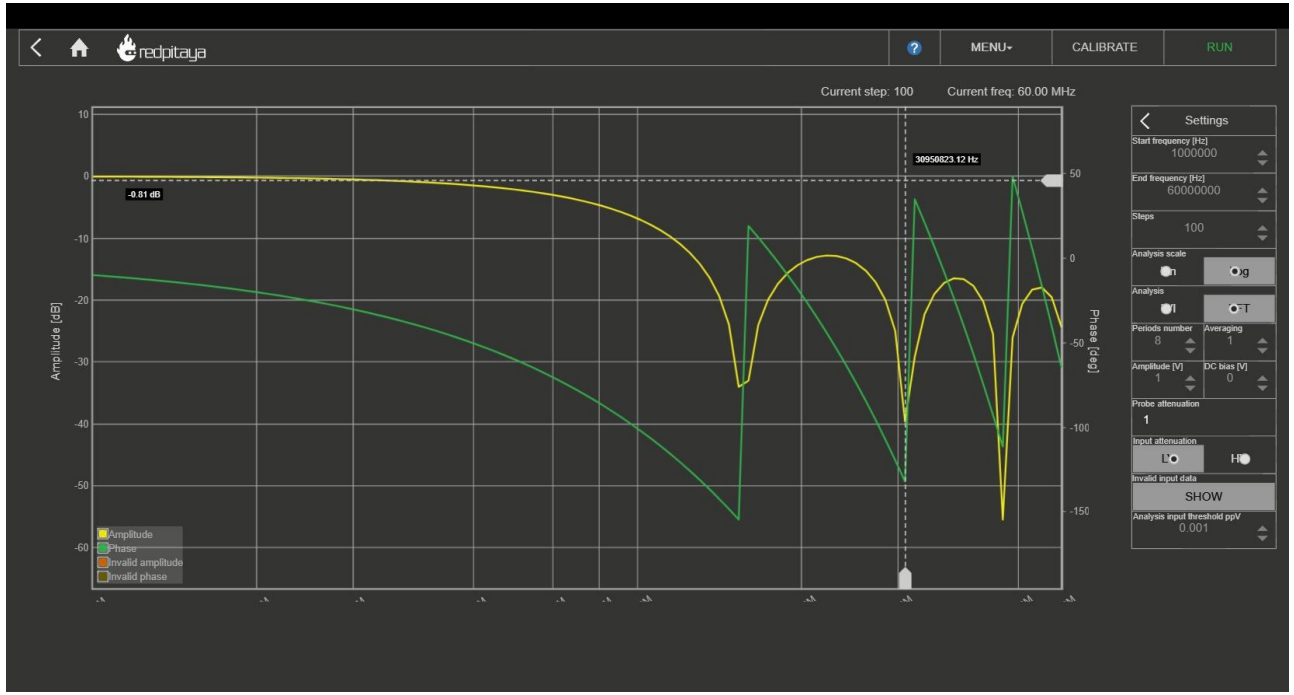
6-5 Test with Web Application

- Generate sine wave: **1 MHz**, 0.5 V at OUT1
  - observe IN1, OUT1 and IN2 (vertical setting 500 mV)
- change frequency to 10 MHz and observe IN2
  - you should notice amplitude decrease
  - even more for 16 MHz or with square wave



6-6 Test With Bode Analyzer

- Run **Bode Analyzer** from the Red Pitaya web applications
- Using **terminal** set data_loop: monitor 0x4000000C 1
- Use Settings to define observed frequency range (1 MHz to 60 MHz), number of sample points and run analysis



High-Level FPGA Design

Lab 7: HLS Design of FIR Filter

In this lab you will:

- Design a low pass Finite Impulse Response (FIR) filter
- Simulate FIR response using different data types
- Synthesize circuits with streaming interface
- Prepare the FIR design for IP integration (required for Red Pitaya)

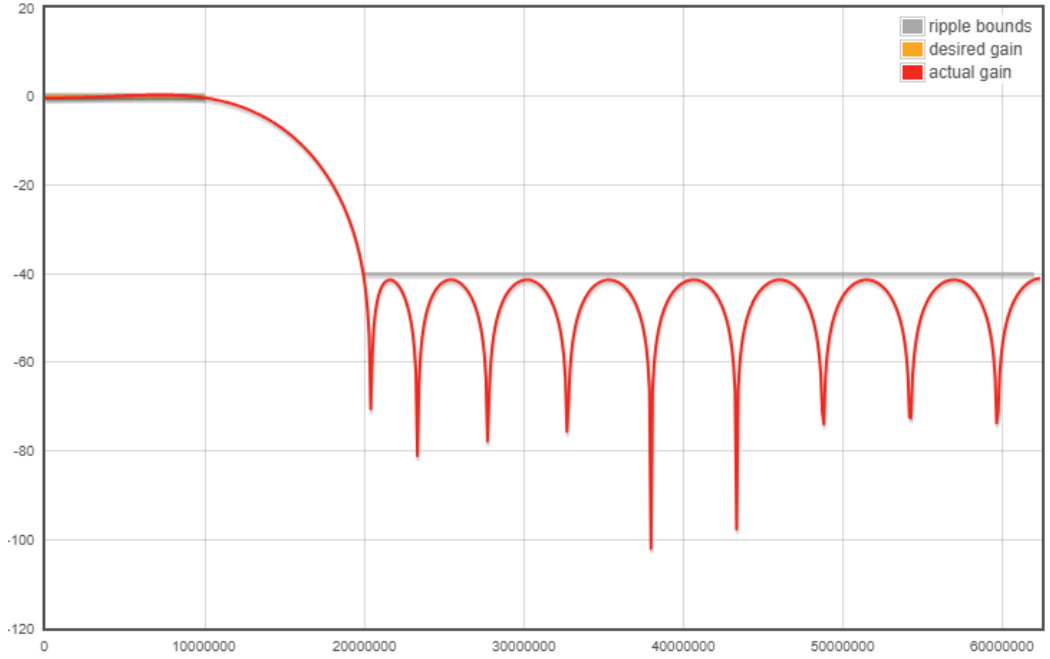
Finite Response (FIR) Filter Design Tools

- FIR Filter Design: https://www.arc.id.au/FilterDesign.html
- FIIIR!: https://fiiir.com/
- TFilter: http://t-filter.engineerjs.com/

$$y[t] = \sum_{i=0}^{N-1} c[i] * x[t-i]$$

Demo FIR Specs

- Sample rate: 125 MHz
- Low Pass FIR, 23 taps
  - passband 0 – 10 MHz, max. ripple 1 dB,
  - transition band: 10 MHz – 20 MHz
  - stopband: 20 MHz – 62.5 MHz, attenuation: -40 dB




```
C[0-11] =
 0.00800359,
 0.00654945,
 0.00047600,
-0.01343992,
-0.02990277,
-0.03805095,
-0.02525160,
 0.01549010,
 0.08019417,
 0.15248278,
 0.20948838,
 0.23108817,
```

Lab Sources

- Project folder structure

```
proj/
  ├── src/
  │     ├── 07_fir/          ← lab4 C++ sources
  │     │     ├── fir.cpp
  │     │     ├── fir.h
  │     │     ├── tb_fir.cpp  ← initial Test Bench
  │     │     └── tb2_fir.cpp
  │
  ├── vitis/              ← Vitis workspace (location)
  │     ├── fir_hls/      ← lab7 HLS component
```

7-1 Create a New Vitis HLS Component

- Create Component…, Create Empty HLS Component
- Component name: **fir_hls**
  Configuration File: Empty File
- Add Source Files: fir.cpp, fir.h
  - Top function: **fir**
- Add only first Test Bench File: tb_fir.cpp
- Part xc7z010clg400-1, clk=8ns

7-2 FIR header

- Data types: DATA_T, COEF_T, ACC_T
- Const: N = 23
- Array of coefficients

```
#define N 23
typedef ap_fixed<14,1,AP_RND,AP_SAT> DATA_T;
typedef ap_fixed<14,1> COEF_T;
static const COEF_T coef_r[N] = {
  0.00800359,
  0.00654945,
  0.00047600,
 -0.01343992,
 -0.02990277,
 -0.03805095,
 -0.02525160,
  0.01549010,
  0.08019417,
  0.15248278,
  0.20948838,
  0.23108817,
  0.20948838,
  0.15248278,
  0.08019417,
  0.01549010,
 -0.02525160,
 -0.03805095,
 -0.02990277,
 -0.01343992,
  0.00047600,
  0.00654945,
  0.00800359
 };
```

7-2 FIR Function

- Use streaming interface
- Add shift and multiply-accumulate loop

$$y[t] = \sum_{i=0}^{N-1} c[i] * x[t-i]$$

```
#include "fir.h"

void fir(
    hls::stream<DATA_T> &in,
    hls::stream<DATA_T> &out
)
{
 static DATA_T shift_r[N];
 ACC_T acc = 0;

 while (!in.empty()) {
    DATA_T x = in.read();
    DATA_T acc = 0;

    // add your FIR logic
    out.write(acc);
 }
}
```

7-3 Simulation: Impulse response

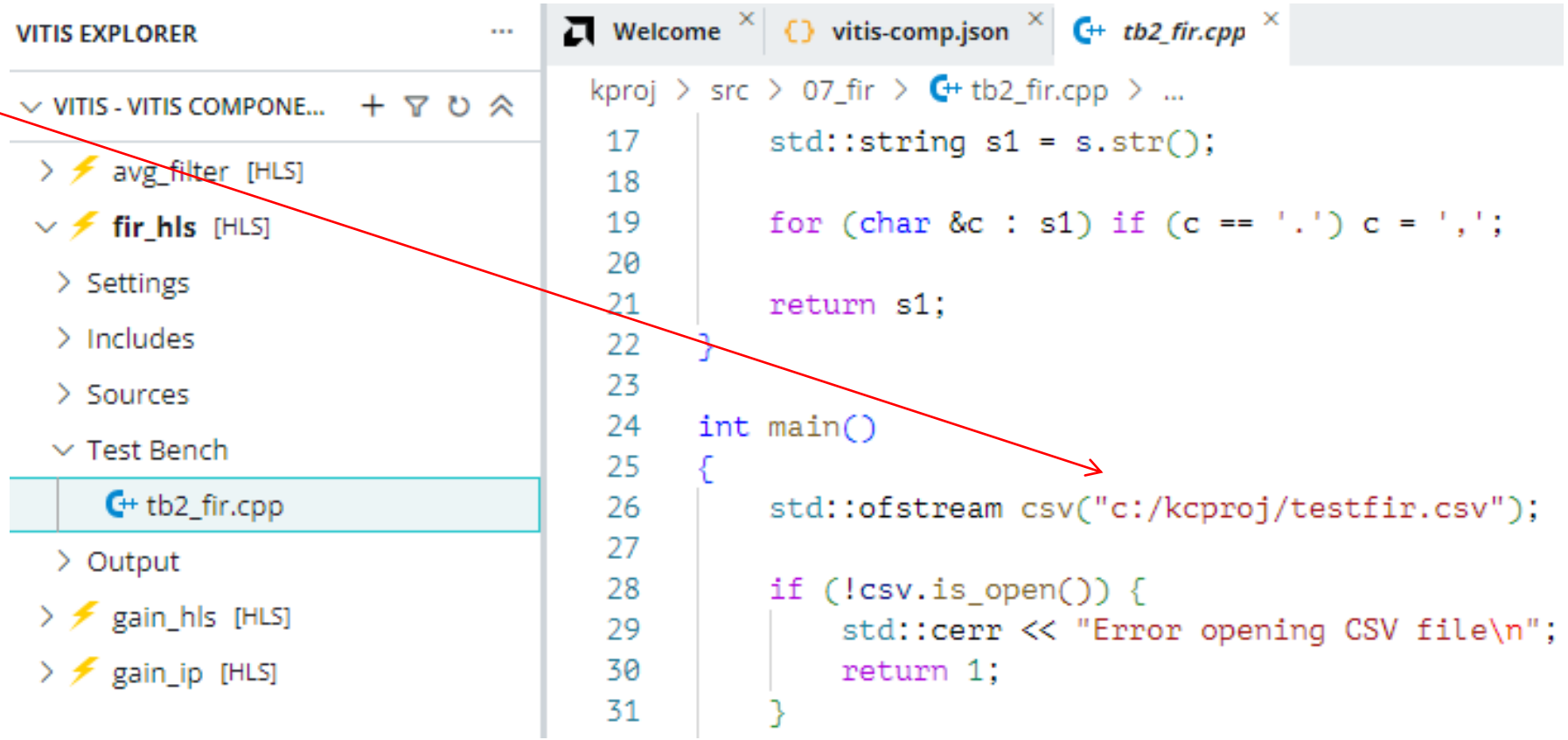
- Test Bench generates impulse
- Verify expected output
  - compare with coefficients

```
========================
Impulse response:
[TB] 0.00793457
[TB] 0.00646973
[TB] 0.000366211
[TB] -0.0135498
[TB] -0.0299072
[TB] -0.0380859
[TB] -0.0252686
[TB] 0.0153809
[TB] 0.0800781
[TB] 0.152466
[TB] 0.209473
[TB] 0.231079
```

```
static const COEF_T coef_r[N] = {
  0.00800359,
  0.00654945,
  0.00047600,
 -0.01343992,
 -0.02990277,
 -0.03805095,
 -0.02525160,
  0.01549010,
  0.08019417,
  0.15248278,
  0.20948838,
  0.23108817,
  0.20948838,
  0.15248278,
  0.08019417,
  0.01549010,
 -0.02525160,
 -0.03805095,
 -0.02990277,
 -0.01343992,
  0.00047600,
  0.00654945,
  0.00800359
 };
```

7-3 Simulation: Sine Sweep

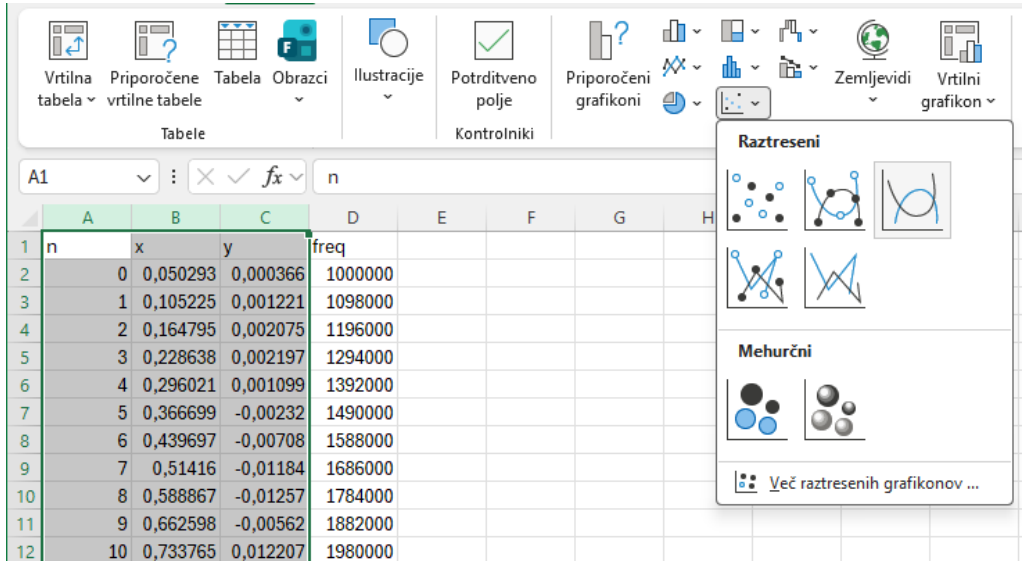
- Delete current and add New Test Bench tb2_fir.cpp
- update output file path
- Run Simulation
- Open output testfir.csv with Excel



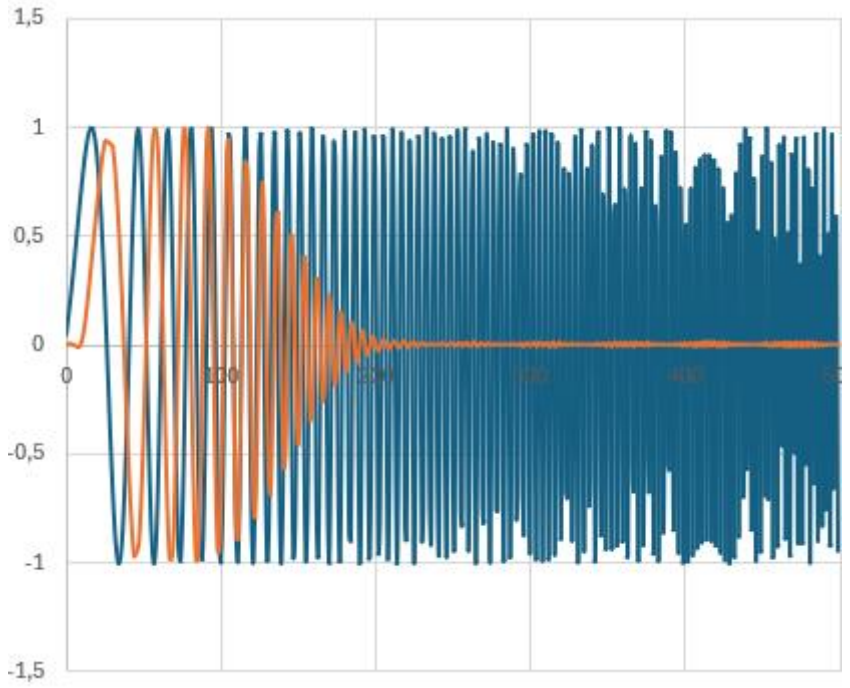
7-3 Simulation: Sine Sweep

Excel

- select first three data columns and display scatter chart

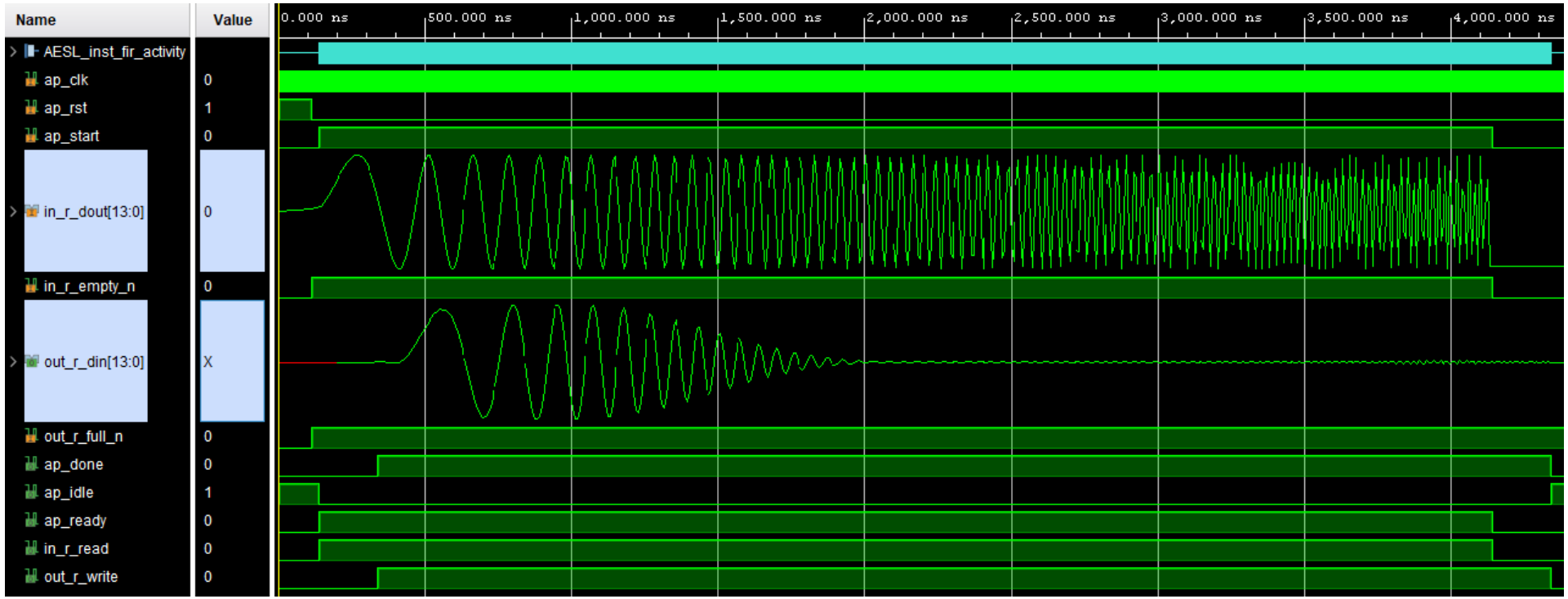


	A	B	C	D
1	n	x	y	freq
2	0	0,050293	0,000366	1000000
3	1	0,105225	0,001221	1098000
4	2	0,164795	0,002075	1196000
5	3	0,228638	0,002197	1294000
6	4	0,296021	0,001099	1392000
7	5	0,366699	-0,00232	1490000
8	6	0,439697	-0,00708	1588000
9	7	0,51416	-0,01184	1686000
10	8	0,588867	-0,01257	1784000
11	9	0,662598	-0,00562	1882000
12	10	0,733765	0,012207	1980000



7-4 C/RTL Cosimulation

- Run **Cosimulation**, set: cosim.trace_level = port
- From **REPORTS** open **Wave Viewer**
- Observe circuit interfacecontrol signals and input and output data
  - set in_r and out_r Radix to Decimal and Format Analog



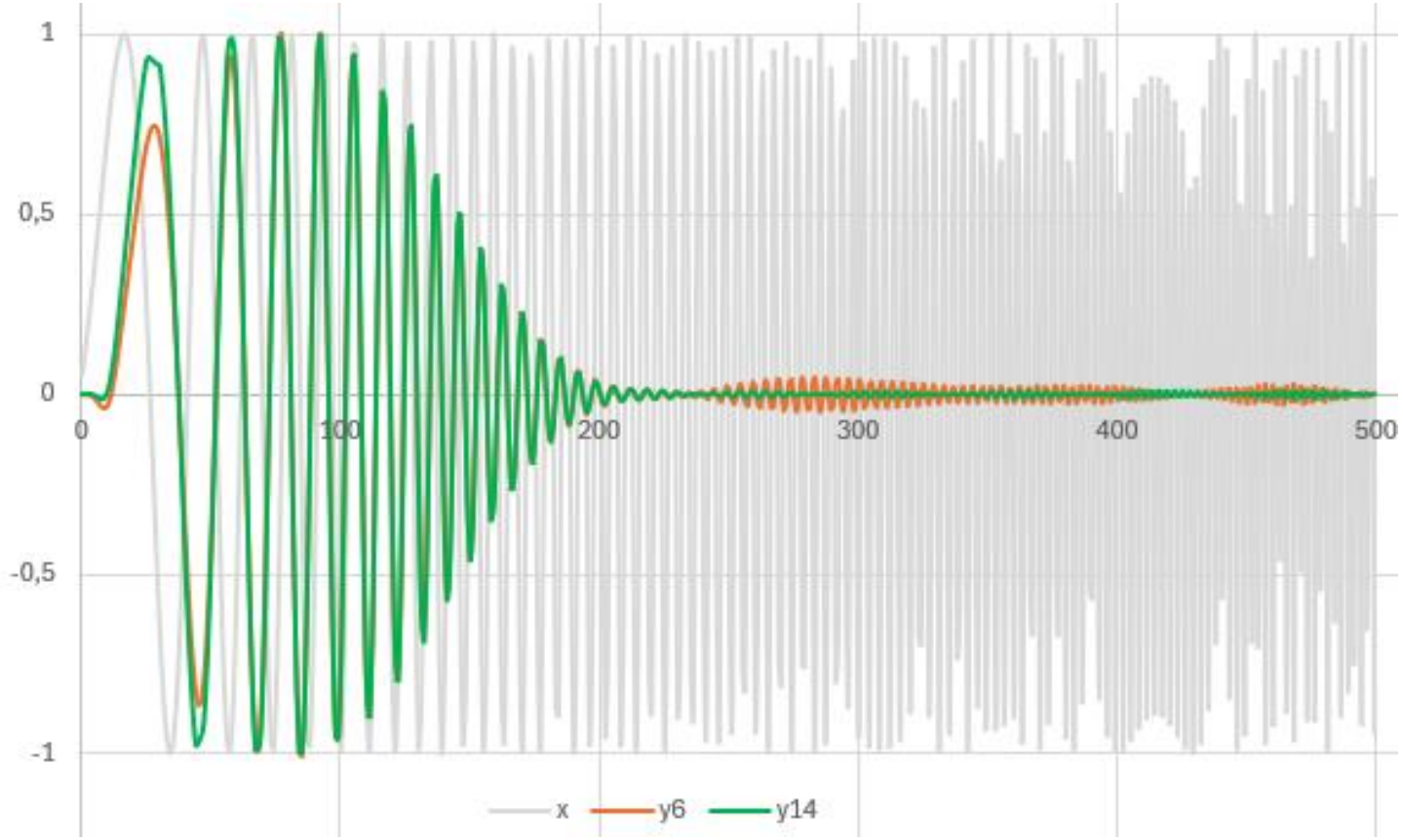
7-5 Design Space Exploration

- Run **C Synthesis** and check Performance & Resource Estimates
- Add directives **UNROLL** on both loops and pipeline on function fir()
- Change coefficients data type COEF_T to 6-bit fixed point

Circuit	Latency	Interval	DSP	FF	LUT
fir, no directives	32	1	17	1905	2835
fir, pipelined					
fir, pipelined, **6-bit COEF_T**					

7-5 Design Space Exploration

- Run C **Simulation** with different fixed point and Verify:
  - Output performance compared to initial design
  - Note higher ripple when less bits are used for coefficients



7-6 IP Component

- Add directive: INTERFACE, ap_ctrl_none
- Run C Synthesis
- Run PACKAGE
  - set output file: **fir_ip** and description: HLS FIR component

*7-7 FIR with Variable Coefficients

- Make a new HLS component
  - right click hls_fir in VITIS EXPLORER and Clone Component, name: hls_fir2
- Edit fir.h
  - make declaration of coefficients as a function parameter
  - COEF_T should be 8-bit values, rename variable to coef

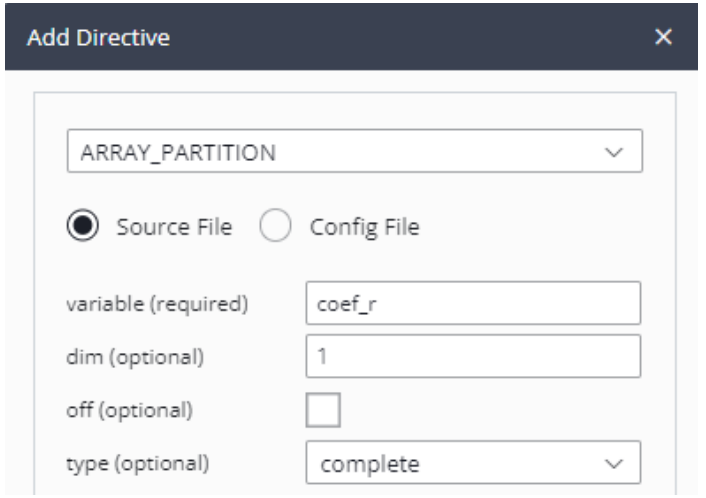
```
typedef ap_fixed<14,1,AP_RND,AP_SAT> DATA_T;
typedef ap_fixed<8,1> COEF_T;
typedef ap_fixed<14,1,AP_RND,AP_SAT> ACC_T;

void fir(
    const COEF_T coef_r[N],
    hls::stream<DATA_T> &in,
    hls::stream<DATA_T> &out
);

const COEF_T coef[N] = {
  0.00800359,
  0.00654945,
  0.00047600,
 ...
```

*7-7 FIR with Variable Coefficients

- Update function declaration in fir.cpp
- Update fir() function call in the Testbench tb2_fir.cpp (line 69)
  `fir(coef, in_stream, out_stream);`
- Run C **Simulation** to Verify the changes
- Run **C Synthesis**
  - Observe HW Interfaces, HLS used dual-port ROM interface to access an array of coefficients
  - Sequential ROM access pipelined prevents FIR operation with interval = 1
- Add INTERFACE **Directive** to parameter coef_r, define mode: ap_memory, storage_type: rom_1p
- Run **C Synthesis**

*7-7 FIR with Variable Coefficients

- Remove INTERFACE **Directive**
- add ARRAY_PARTITION Directive to parameter coef_r
- Run C Synthesis
  - Observe P&R Estimates and HW interfaces
  - The circuit has 24 ports for coefficients and is able to run pipelined FIR



High-Level FPGA Design

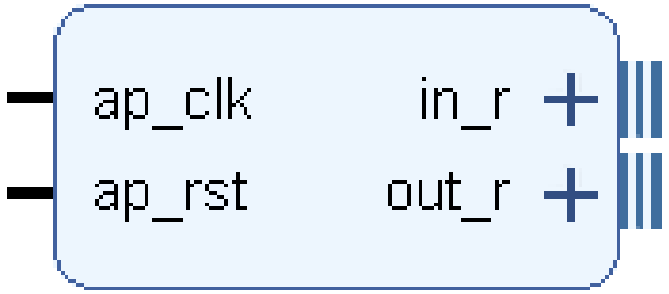
Lab 8: RedPitaya Project with FIR Filter

In this lab you will:

- Include FIR filter IP to the RedPitaya Vivado project
- Verify FIR operation on RedPitaya using Bode Analyser

8-1 Add FIR IP to Red Pitaya Project

- Open the project from Lab 6
- Locate HLS FIR filter IP component folder
- Use Vivado **Project Settings** add hls_fir repository
- Add **Fir** IP to the Project



8-2 Modify proc.sv

- Open proc.sv and change avg_filter instance to fir_0
- Modify signal connections to match streaming interface:
  - in_r_empty_n connected to 1 (always ready data source)
  - out_r_full_n connected to 1 (always ready sink)

```
fir_0 i_filt (
  .ap_clk        (clk_i),
  .ap_rst        (~rstn_i),
  .in_r_dout     (gain_o),
  .in_r_empty_n  (1'b1),
  .out_r_din     (filt_o),
  .out_r_full_n  (1'b1)
 );
```

8-3 Generate Bitstream and Upload

- Generate Bitstream
  - Runs Synthesis, Implementation and Generate Bitstream
- In TCL Console change directory to your project folder and run tcl script `source ./format.tcl` producing final **proc.bit.bin**
- upload **proc.bit.bin** to Red Pitaya
- In Bitwise SSH Client Open New terminal console
  - run ./set.sh proc.bit.bin
- Open web browser at http://192.168.1.15/ and click **Bode Analyser**
- Using **terminal** set data_loop: monitor 0x4000000C 1
- Run Analysis

8-4 FIR Frequency response