Red Pitaya

Thesis

Raphael Frey
Noah Hüsser

July 30, 2017
Version 0.0.1

Contents

List of Figures

List of Tables

List of Listings

CHAPTER 1

Introduction

Measuring equipments has ever been very expensive. If one wants precise devices it can go up to hundreds or thousands of Francs. Modern, cheap FPGAs can often, if combined with a proper frontend, replace those expensive dedicated devices. The frontend consists of dedicated ADCs and DACs and oftentimes some analog filters. Those chips also have become a lot cheaper in recent years aand are thus way more accessible. Extreme equipment, for example with high sampling rates, in contrary can still not be replaced easily. This project aims at arming an FPGA board with logic that can record, filter and store electrical signals with adjustable sampling rates up to 125 MHz. To complement the hardware part a software that runs on an embedded Linux on the integrated ARM core will be coded such that it can transmit the recorded samples over the network. To read and visualize the samples at the other end of the network, another piece of software will be crafted, that will run on the users computer. The primary focus lies on enabling students to analyze audio signals. Since audio signals contain very low frequencyies only up to tens of thousands of Hz they can be sampled with rather low frequencies and thus making FPGAs an excellent choice. An FPGA not only shines in price competitiveness but also in flexibility. This means that the logic is not fixed in silicon and can be adjusted after the product has already been delivered.

For this project a RedPitaya board is used. It is ideal since it features a fast (125 MHz) 14-bit ADC. This poses a huge amount of data, that is not even required for audio signals. Furthermore it is not realisticly possible to transmit this huge amount of data over the network.

Thus the first primary target of this thesis is to decimate the recorded signals. To avoid aliasing effects that emmerge when decimating a signal appropriate filters have to be designed and impemented that are able to attenuate unwanted signal frequencies.

The second primary target of this thesis is the design and implementation of a software-based oscilloscope. This is a graphical user interface that communicates with the RedPitaya board and visualizes the recored samples. Traditional measuring equipment always has a built in display that visualizes the data on the device itself. This uses up a lot of space and provides very low flexibility. Since it can be assumed that every engineur is equipped with a computer, said device should be used to display the signals. This keeps cost and required space down. The data is then transmitted via the network which will be interfaced by both the users computer and the RedPitaya board.°

- Rationale (Why?)
- What is the general approach to solve this problem?
- What has been done so far?
- Results of previous work
- What are we going to do?
- What are the contents of this report?

[**?**]

CHAPTER 2

Theoretical Background

2.1 Analog-to-Digital Data Acquisition

2.1.1 The DSP Chain

Digitally acquiring a signal generally requires at least the following steps:

- Analog LP: Remove any frequencies above fs/2, to enable correct processing down the chain (more on this later).
- ADC: convert time-continuous and value-continuous into value-discreet and time-discreet signal. Being value-discreet is of less importance in our system, but let it be mentioned that this is the source of the quantization noise. Being time-discreet has a few more consequences:
  - multiplication w/ dirac pulse sequence in time domain
  - convolution w/ dirac pulse sequence in frequency domain
  - spectrum of signal is repeated at intervals of fs, centered around each multiple of fs.
  - if the signal has frequency components above fs/2, this means that different copies of the signal's spectrum will overlap and lead to an error called aliasing. This means that the analog waveform can no longer be unambiguously reconstructed and correct processing of the data down the chain is not possible. The phenomenon called "folding back" is also a consequence of this and will be described in more detail later, because it is of particular importance for our system.
  - As a consequences, the analog low-pass filter is required.
- DSP: Can be n inalmost arbitrary. Preferred to analog signal processing because it can be done with computers, which are well understood and cheap. Primary problem tends to be the amount of data being shoved into the DSP system, which is usually constrained with regards to its available resources (i.e. processing power). If the ADC provides too much data for the DSP to meaningfully process (keep in mind: depending on *what* is to be done in the DSP part, sometimes the ADC's data rate might be too high, other times not, so it is usually not possible to perfectly match

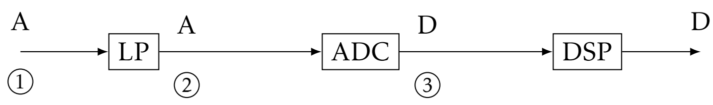

Figure 2.1: Digital signal processing chain from analog signal to the digitally processed data stream

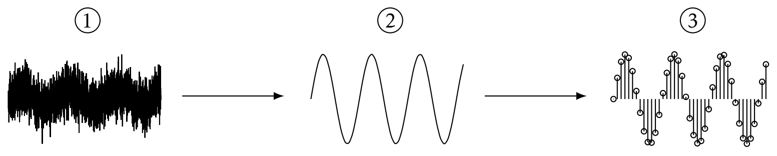

Figure 2.2: Digital signal processing chain from analog signal to the digitally processed data stream

the ADC's specifications to the DSP unless the application's scope is very narrowly defined before the system is designed).

The solution to the problem of too much data is usually downsampling. One could run the ADC at lower clock frequencies, but oversampling carries some advantages which would be lost with that.

2.1.2 Challenges in Downsampling

The most obvious way to downsample from a sequence of values is of course to simply pick each nth sample. However, this has some serious drawbacks which make it an unworkable solution in most cases.

Fancy Graphics of downsampling without LP filter with explanations

2.1.3 Digital Low-Pass Filters

The obvious solution to this predicament is to apply a (digital) low-pass filter to the sequence of values before downsampling. For this purpose, three types of filters are commonly used, each with their own specific advantages and drawbacks: IIR, FIR, CIC.

For theseandthose reasons, we will use FIR and CIC in our system.

Fancy graphics of LP filter, downsampling and folding back

2.2 Designing a Filter System

Talking about which type of filter has which properties is all good and well in theory, but how does one actually apply this knowledge to a practical problem? This section answers that question insofar as it applies to our project.

- limited HW resources
- single-stage vs. multi-stage
- TBW issue with multi-stage
- filters at lower frequencies use fewer resources
- halfband filtres
- CIC: compensation filters

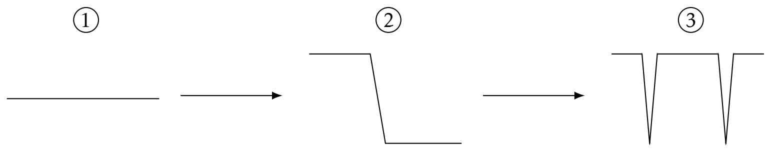

Figure 2.3: Digital signal processing chain from analog signal to the digitally processed data stream

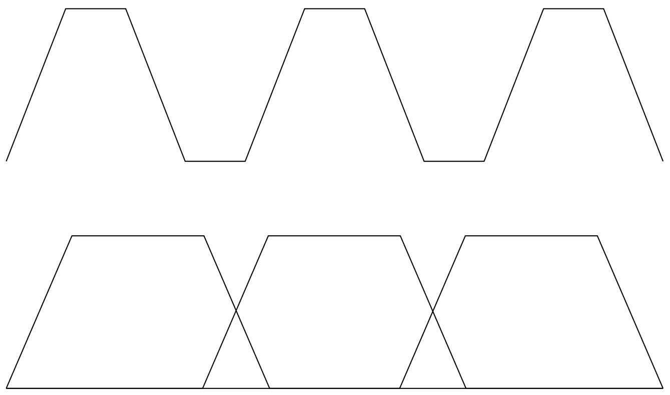

Figure 2.4: Digital signal processing chain from analog signal to the digitally processed data stream

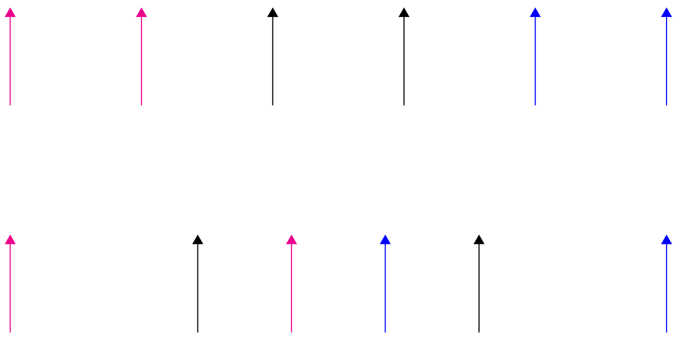

Figure 2.5: Digital signal processing chain from analog signal to the digitally processed data stream

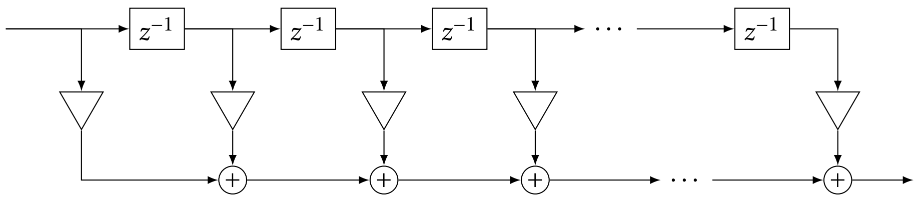

Figure 2.6: Digital signal processing chain from analog signal to the digitally processed data stream

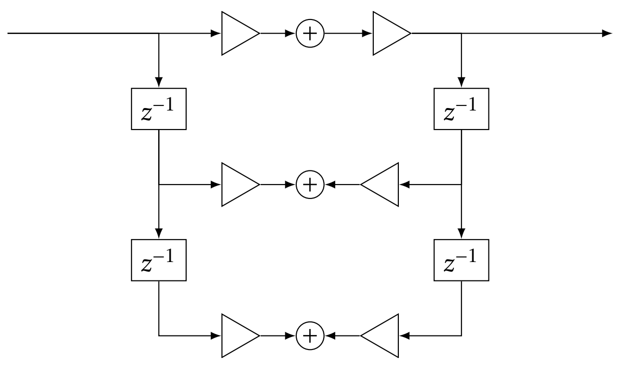

Figure 2.7: Digital signal processing chain from analog signal to the digitally processed data stream

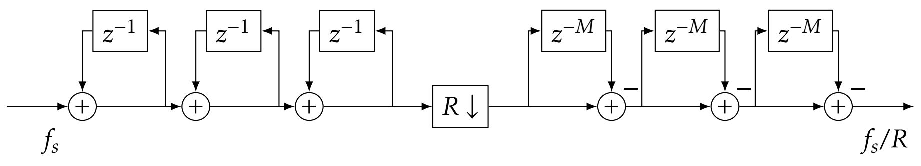

Figure 2.8: Digital signal processing chain from analog signal to the digitally processed data stream

Part I

Implementation

Implementation can be read independently of previous part, but there should be a red thread from decision to . Deals primarily with design decisions.

Present a diagram with all system components. Then document the components in their respective chapters and sections.

CHAPTER 3

Data Acquisition System

3.1 FPGA

3.2 Kernel Module

CHAPTER 4

Filters

CHAPTER 5

Server

CHAPTER 6

Graphical Front End

To view measured data a graphical user interface (GUI in further text) was created. It can receive the recorded samples over the network and display them on a canvas. Furthermore it manages triggers and does a lot of math to get more specific metrics of a signal. In this section the requirements for this piece of software, the design choices and the implementation details are discussed.

6.1 Requirements

The requirements for the GUI were given by the scope of Prof. Gut's 'Spektrum Analyzer' written in Java. The task description required the new GUI to have the same features as the old one plus as many more as possible.

The requirements were as follows

- Receive data over the network.
- Display received data in time as well as fequency space.
- Calculate the RMS power density in the signal.
- Calculate the THD ratio of the signal.

6.2 Design Choices

There is a wealth of programming languages to choose from. And there are as many libraries helping with graphics and networking as well for most of those languages. In the following it is explained why we chose JavaScript and web technologies to implement a basic GUI.

In the comparison matrix 6.1 a select few popular possibilities were given weights for certain attributes of the respective language.

All the attributes are explained in the following.

Open Standard

Since this is a university based project meant for educational purposes too, it was very important to make all source code available under public license. Thus it was important to have a company and paid model independant sulution. Many languages are managed by a council or similar and open to public commits and thus deemed an open standard. Some are managed by a company and not classiefied as a open standard.

Networking

To ensure a fast and lossless data transfer, it was very important to have the choice between good networking protocols as well as convenient libraries to ease the use of those standards. Networking is not a trivial thing and standards can be quite engineering and feature heavy. Thus it is important to have ready-to-use libraries that abstract the network. For more information on evaluated networking solutions, read Section **??**.

Graphics

An oscilloscope is quite requiring when it comes to graphics, since a image-stream that is fluent for the human eye has to be provided in a high resolution. This fact made it indispensable to use a library that interfaces OpenGL. Since a interface is not easy to design from scratch only using rectangles and circles, it was deemed important to have a GUI toolkit that makes the design process of the GUI easy. More on possible graphics libraries in section **??**

Widespread

It was important for the project to use a widespread solution since that way it is easy to obtain help and ask more savy users about certain pitfalls.

User-Friendly

Some solutions are more user-friendly when it comes to toolkit and usage. Since both team members come form a Linux background, it was strongly preferred to use a language that does not quasi-require a huge IDE or requires a lot of uneasy maintenance.

Easy To Use(r)

Since not all users want to fight with installers and package managers, the deployment options as well as general stability of the environment for the binaries were a strong point in the descision process.

Familiarity With The Language

The best toolkits do not matter if none of the involved programmers have ever used it and will struggle with even the basics for a major part of the project duration. Thus it was unavoidable to have some personal preferences for some languages.

After weighting in all the different aspects JavaScript was chosen as the language to implement the GUI for the oscilloscope. JavaScript is a scripting language that can be interpreted by the browser. It is known for it's high versatility and widespread use in the web community. A few years ago, JavaScript would not have been a viable choice for graphics and networking at all. But with the recent addition and more important great increase in

	Rust	C++	Java	Python	JavaScript
Open Standard	6	6	1	6	6
Networking	6	6	6	6	4
Graphics	2	5	5	5	6
Widespread	3	6	6	5	6
User-Friendly	5	5	5	5	6
Easy To Use(r)	3	4	5	6	6
Familiarity With The Language	3	4	3	6	6
Total	28	36	31	39	40

Table 6.1: Weights of certain aspects of possible programming languages.

stability in performance of WebGL and WebSockets, JavaScript has become a very potent solution available to everyone. With JavaScript deploying the application to the enduser is as simple as making it accessible via a website that runs on the RedPitaya board. Thus it is very convenient for the user to work with the board, considering the assumption that every user has a webbrowser that is able to run the application. A downside of JavaScript is the huge runtime the browser needs to execute the application and thus resulting in a lot of memory ressources used. Since those are easily available nowadays, this issue was considered non-relevant. Another downside of JavaScript is that it leaves very little room when it comes to networking choices. For streaming data there is only WebSockets that performs well. Since WebSockets is a quite capable solution, this issue was also weighted rather light.

6.2.1 Networking

To ensure a fluent stream of data, very little overhead for the transmitted data is key.

Normally for streamed data where packets can be lost, **UDP** is the best choice since it has no overhead for guaranteeing completeness and in-order for all packets sent resulting in a packet header of only 2 bytes [**?**]. UDP sends packets but does not guarantee that none are lost. Since the scope only requires complete frames, those could be transmitted with a size of one UDP packet and thus ensuring no sample in the frame is lost. If no-lost-packets should be guaranteed, that would have been to implement.

For guaranteed transmission and sequentiality of the data, one is advised to use **TCP**. This comes at the cost of some more overhead resulting in a 6 byte header [**?**]. Considering the huge packet size this header is, with less than 0.1 % of the total packet size, completely negligible. What TCP does more and was important for this project is congestion control. It ensures that no more packets are sent if old ones are missing. It prevents the network from collapsing because an UDP sender sends all packets it can and thus using the entire bandwidth even if the receiver cannot even process the data at this point. This means that TCP also helps when the bandwith is small by waiting for the current package and not already sending further packages and thus providing sort of automatic bandwith adaption. There is the possibility of using raw TCP sockets or one of TCP's subprotocols. Raw sockets require the user to implement their own protocol entirely to handle data transmission on an application layer whilst using subprotocols already provide a standard way to do so.

Two of those subprotocols are HTTP and WebSockets. HTTP comes with great overhead and is meant for single transactions only. WebSockets on the contrary are meant for data streaming. Since JavaScript enforces WebSockets the functionality is explained a little bit more in the following.

6.2.1.1 WebSockets

WebSockets final RFC 6455[**?**, **?**] was released in December 2011 and is thus still quite young. It is meant to compensate the lack of raw UDP and TCP sockets in JavaScript which is due to security threats that are not further elaborated here. WebSockets is located in the Application Layer of the OSI model[1]. Instead of opening a raw WebSocket, the handshake is done via HTTP(S). This brings the benefit of communicating through the same ports as the browser (80 or 443) which enables the protocol to go through most firewalls. The client sends an upgrade request to the server which then opens a WebSocket connection. This allows for a very conventient way to use TCP Sockets without any entirely new standards. The section "1.5 Design Philosophy" in the RFC 6455[**?**, **?**] explains it very well: Basically it is intended to be as close to just exposing raw TCP to script as possible given the constraints of the Web.

The only exception is that WebSockets adds framing to make it packet rather than stream based and to differentiate between binary and text data. This differentiation is very useful for this project. Instructions to the server are issued via the text channel whilst data is sent back through the binary channel, allowing for very convenient interfacing with close to no effort.

6.2.2 Graphics

The graphics portion of the GUI is the most important part. Since the GUI should plot data fast and conveniently as well as display some numbers and provide controls to manipulate the view, it is important to have a good library, that enables all those things. It is absolutely key to render the graphics on the GPU. Since the application should be cross platform and open source, libraries using OpenGL are a good choice.

With the choice of JavaScript & HTML there comes a great wealth of libraries that enable the user to easily write GUI applications. Prototyping is fast and with CSS and a lot of different frameworks the GUI is also nice-looking.

A few frameworks were evaluated to build the controls of the scope, with mithril.js finally being chosen for it's simplicity and flexibility. Mithril is a framework with an exceptionally low footprint and high DOM recalculation. Those two facts are key to a good WebUI, since the User does not want to load lots of data and also does not want to experience any lag when building up the UI again. More on mithril.js in section **??**.

To plot the data some graphing libs could have been used. Those would namely be plotly.js or chart.js. Whilst they bring in a lot of built in functionality like logarithmic plots or automatic axis labeling, they also have a quite heavy overhead. Practical experience and tests have shown that both of them are not meant and performant enough to plot high amounts of data in real time. Thus it was decided to use WebGL draw calls directly to draw on a HTML canvas. A HTML canvas is an environment that is directly exposed from the GPU to the user such that he can use GPU rendering inside the browser. More on WebGL and it's functioning in section **??**

[1]TODO: https://en.wikipedia.org/wiki/OSI_model

6.3 Implementation

6.3.1 WebGl

TODO: how do we work with webgl (sample draw calls, important callbacks, code-samples)

6.3.2 mithril.js

TODO: how do we work with mithril.js (basic concepts, important to know, code-samples)

6.3.3 WebSockets

TODO: how do we work with websockets (important callbacks, code-samples)

6.3.4 Application Structure

TODO: basic structure of the application

6.3.5 Power Calculation

TODO: how do we calculate rms power and the power density spectrum

6.3.6 SNR Autodetection

TODO: how do we calculate the snr (incl. windowing etc)

6.3.7 THD Calculation

TODO: how do we calculate THD

6.4 Product

TODO: images, features, etc

Part II

Developer Guide

Documentation for a person who wishes to utilize our system in their work and/or improve upon it?

Make sure to distinguish between *Implementation* and this part. Lines seem a bit blurry to me (R.F.) at the moment (July 30, 2017).

CHAPTER 7

Project Structure

Structure of the repository. What can be found where, and what to do with it?

CHAPTER 8

IP Core

Documentation of our FPGA Project (structure, interfaces, registers . . .)

CHAPTER 9
Linux

Kernel module, server

CHAPTER 10

Tool Chain

Vivado, Build Box, ARM Linux, TCL, Makefiles, Libs for building server application

Part III

User Guide

Documentation for the end user. Primarily concerned with the front-end.

Appendices

APPENDIX A

Code Listings

Shell commands can be type thusly:

```
user:> if [ -f "${myfile}" ];then echo "${myfile} exists!"
```

A.1 Makefile

Listing A.1: Makefile Code

```
# -------------------------------------------------------------------------------
# General constants

# Change to adjust the output directory
BUILD = build

# -------------------------------------------------------------------------------
# Constants for the FPGA Core

VIVADO = vivado -nolog -nojournal -mode batch
PART = xc7z010clg400-1

all: all-cores zynq_logger project

all-cores:
	$(VIVADO) -source create_cores.tcl -tclargs $(PART) $(BUILD)/cores
	rm -f vivado*
	rm -f webtalk*

axis_to_data_lanes:
	$(VIVADO) -source create_cores.tcl -tclargs $(PART) $(BUILD)/cores axis_to_data_lanes_v1_0
	rm -f vivado*
	rm -f webtalk*

.PHONY: zynq_logger
zynq_logger:
	cd zynq_logger && make core

project:
	$(VIVADO) -source make_project.tcl
	rm -f vivado*
	rm -f webtalk*

clean:
	rm -rf $(BUILD)
	rm -rf .Xil
	rm -rf .tmp_versions
```

A.2 Verilog

Listing A.2: Verilog Code

```
timescale 1 ns / 1 ps

module axi_axis_reader #
(
  parameter integer AXI_DATA_WIDTH = 32,
  parameter integer AXI_ADDR_WIDTH = 16
)
(
  // System signals
  input  wire                        aclk,
  input  wire                        aresetn,

  // Slave side
  input  wire [AXI_ADDR_WIDTH-1:0] s_axi_awaddr,  // AXI4-Lite slave: Write address
  input  wire                        s_axi_awvalid, // AXI4-Lite slave: Write address valid
  output wire                        s_axi_awready, // AXI4-Lite slave: Write address ready
  input  wire [AXI_DATA_WIDTH-1:0] s_axi_wdata,   // AXI4-Lite slave: Write data
  input  wire                        s_axi_wvalid,  // AXI4-Lite slave: Write data valid
  output wire                        s_axi_wready,  // AXI4-Lite slave: Write data ready
  output wire [1:0]                  s_axi_bresp,   // AXI4-Lite slave: Write response
  output wire                        s_axi_bvalid,  // AXI4-Lite slave: Write response valid
  input  wire                        s_axi_bready,  // AXI4-Lite slave: Write response ready
  input  wire [AXI_ADDR_WIDTH-1:0] s_axi_araddr,  // AXI4-Lite slave: Read address
  input  wire                        s_axi_arvalid, // AXI4-Lite slave: Read address valid
  output wire                        s_axi_arready, // AXI4-Lite slave: Read address ready
  output wire [AXI_DATA_WIDTH-1:0] s_axi_rdata,   // AXI4-Lite slave: Read data
  output wire [1:0]                  s_axi_rresp,   // AXI4-Lite slave: Read data response
  output wire                        s_axi_rvalid,  // AXI4-Lite slave: Read data valid
```

```
    input  wire                      s_axi_rready,  // AXI4-Lite slave: Read data ready

    // Slave side
    output wire                      s_axis_tready,
    input  wire [AXI_DATA_WIDTH-1:0] s_axis_tdata,
    input  wire                      s_axis_tvalid
  );

    reg int_rvalid_reg, int_rvalid_next;
    reg [AXI_DATA_WIDTH-1:0] int_rdata_reg, int_rdata_next;

    always @(posedge aclk)
    begin
      if(~aresetn)
      begin
        int_rvalid_reg <= 1'b0;
        int_rdata_reg <= {(AXI_DATA_WIDTH){1'b0}};
      end
      else
      begin
        int_rvalid_reg <= int_rvalid_next;
        int_rdata_reg <= int_rdata_next;
      end
    end

    always @*
    begin
      int_rvalid_next = int_rvalid_reg;
      int_rdata_next = int_rdata_reg;

      if(s_axi_arvalid)
      begin
        int_rvalid_next = 1'b1;
        int_rdata_next = s_axis_tvalid ? s_axis_tdata : {(AXI_DATA_WIDTH){1'b0}};
      end

      if(s_axi_rready & int_rvalid_reg)
      begin
        int_rvalid_next = 1'b0;
      end
    end

    assign s_axi_rresp = 2'd0;

    assign s_axi_arready = 1'b1;
    assign s_axi_rdata = int_rdata_reg;
    assign s_axi_rvalid = int_rvalid_reg;

    assign s_axis_tready = s_axi_rready & int_rvalid_reg;
```

```
endmodule
```

A.3 VHDL

```
----------------------------------------------------------------------------------
--
-- comparator.vhd
--
-- (c) 2015
-- L. Schrittwieser
-- N. Huesser
--
----------------------------------------------------------------------------------
--
-- Old descision piece for the trigger units; obsolete
--
----------------------------------------------------------------------------------


library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;

-- Uncomment the following library declaration if instantiating
-- any Xilinx leaf cells in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity comparator is
        generic (
                Width : integer := 14
        );
    port (
        AxDI : in unsigned(Width - 1 downto 0);
        BxDI : in unsigned(Width - 1 downto 0);
        GreaterxSO : out std_logic;
        EqualxSO : out std_logic;
        LowerxSO : out std_logic
    );
end comparator;

architecture Behavioral of comparator is
begin
    process(AxDI, BxDI)
    begin
        GreaterxSO <= '0';
        EqualxSO <= '0';
        LowerxSO <= '0';
        if AxDI > BxDI then
            GreaterxSO <= '1';
        elsif AxDI = BxDI then
            EqualxSO <= '1';
        else
            LowerxSO <= '1';
        end if;
    end process;
end Behavioral;
```

A.4 TCL

```
# ====================================================================================================
# make_cores.tcl
#
# Simple script for creating and installing all the IPs in a given directory.
# The script must be run from inside the directory it resides.
#
# by Noah Huesser <yatekii@yatekii.ch>
# based on Anton Potocnik, 01.10.2016
# ====================================================================================================

set part_name [lindex $argv 0]
set build_location [lindex $argv 1]
if {[llength $argv] > 2} {
        set core_names [lindex $argv 2]
}

#set cores [lindex $argv 0]
set cores cores

if {$rdi::mode != "batch"} {[puts "Installing cores from $cores into Vivado..."]}

if {! [file exists $cores]} {
        puts "Directory $cores was not found. No cores were installed.";
        return
}

# Generate a the list of IP cores in the $cores directory if we didn't receive names
if {! [info exists core_names]} {
        cd $cores
        set core_names [glob -type d *]
        cd ..
}

#set core_names "axis_to_data_lanes_v1_0";
#set core_names "axis_red_pitaya_adc_v1_0";

# Import Pavel Demin's Red Pitaya cores
foreach core $core_names {
        set argv "$part_name $build_location $core"
        if {$rdi::mode != "batch"} {[puts "Installing $core..."]}
        source scripts/add_core.tcl
        if {$rdi::mode != "batch"} {[puts "=========================="]}
}
```

```
entity comparator is
    generic (
        Width : integer := 14
    );
    port (
        AxDI : in unsigned(Width - 1 downto 0);
        BxDI : in unsigned(Width - 1 downto 0);
        GreaterxSO : out std_logic;
        EqualxSO : out std_logic;
        LowerxSO : out std_logic
    );
end comparator;
```

Listing A.3: Comparator

A.5 Matlab

Here, we shall also demonstrate breaking a code file into multiple segments to comment on its contents.

We start with the header:

Listing A.4: Matlab Code

```
% ----------------------------------------------------------------------------%
% FILTER DESIGN ITERATIONS
%
% DESCRIPTION
% Designs and showcases various filter chains for evaluation.
%
% AUTHORS:
% Raphael Frey, <rmfrey@alpenwasser.net>
%
% DATE:
% 2017-MAY-12
% ----------------------------------------------------------------------------%
```

In the next box, we start where we left off previously, and we pack some more header information into our listing:

```

% Parameter Description
% R: rate decimation
% N: Number of CIC filter stages
% M: differential delay in CIC combs

% Global Input Sampling Frequency: 125 MHz
%
% Desired Target Frequencies:       25 MHz (R =                5)
%                                    5 MHz (R = 5^2       =   25)
%                                    1 MHz (R = 5^3       =  125)
%                                  200 kHz (R = 5^4       =  625)
%                                  100 kHz (R = 5^4 * 2   = 1250)
%                                   50 kHz (R = 5^4 * 2^2 = 2500)
```

Then we describe the target for the FIR filter:

```
%% ========================================================= FIR: Target: 25 MHz
%
% Specify a number of FIR lowpass filters for a permutation of:
% - Start Frequency of the pass band (upper edge)
% - Stop Frequency of the stop band (lower edge)
% - Ripple in pass band
% - Attenuation in stop band
%
%
% See also:
% https://ch.mathworks.com/help/signal/ref/fdesign.lowpass.html
% https://ch.mathworks.com/help/dsp/ref/fdesign.decimator.html
```

After which we start the sript proper by setting up the iteration parameters:

```
clear all;close all;clc;
% ----------------------------------------------- Input Sampling Frequency in Hz
Fs  = 125e6;

% ------------------------------------------------------------ Decimation Factor
R   = 5;

% -------------------------- Frequency at the Start of the Pass Band; Normalized
% NOTE: The smallest number in Fp must be smaller than the smallest number in
%       Fst (see below).
Fp  = [0.1 0.15 0.2];

% --------------------------- Stop band frequencies ("How steep is the filter?")
% NOTE: The smallest number in Fst must be larger than the largest number in
%       Fp (see above).
Fst = [0.21 0.22];

% ----------------------------------------------------- Ripple in Passband in dB
Ap  = [0.25 0.5 1];

% ----------------------------------------------- Attenuation in Stop Band in dB
Ast = [20 40 60 80];
```

We define some data structures to contain the filter objects for further processing:

```
% -------------------------------------------------------- Filter Design Objects
Hd  = cell(length(Fp),length(Ap),length(Fst),length(Ast),2);

% ------------------------------- Lengths of Numerators for the Different Filters
% Saves the length of the numerator for each permutation of Fp, Ap, Fst and Ast,
% along with those parameters themselves. Each filter gets a number as well (see
% 't' below).
NumL  = cell(length(Fp),length(Ap),length(Fst),length(Ast),6);

% -------------- Same Thing, for more convenient extraction to file or somesuch
% Saves the length of the numerator for each permutation of Fp, Ap, Fst and Ast,
% but without those parameters.
NumL2 = [];
```

And then we iterate:

```
% Plot as we proceed. This enables color cycling by default.
figure;hold on;
t = 0;              % total number of filters; filter number
l = 1;              % cell index for Fp
for fp = Fp
    i = 1;          % cell index for Ap
    for ap = Ap
        j = 1;      % cell index for Fst
        for fst = Fst
            k = 1; % cell index for Ast
            for ast = Ast
                d = fdesign.decimator(...
                    R,...
                    'lowpass',...
                    'Fp,Fst,Ap,Ast',...
                    fp,...
                    fst,...
                    ap,...
                    ast);

                Hd{l,i,j,k,1} = design(d,'SystemObject',true);
                Hd{l,i,j,k,2} = t;

                NumL{l,i,j,k,1} = fp;
                NumL{l,i,j,k,2} = ap;
                NumL{l,i,j,k,3} = fst;
                NumL{l,i,j,k,4} = ast;
                NumL{l,i,j,k,5} = t;
                NumL{l,i,j,k,6} = length(Hd{l,i,j,k,1}.Numerator);
                NumL2 = [NumL2 NumL{l,i,j,k,6}];
                scatter(t,NumL{l,i,j,k,6});
                k = k+1;
                t = t+1;
            end
            j = j+1;
        end
        i = i+1;
    end
    l = l+1;
end
% hfvt = fvtool(Hd{:,:,:,:,1},'ShowReference','off','Fs',[Fs]);
```

Index