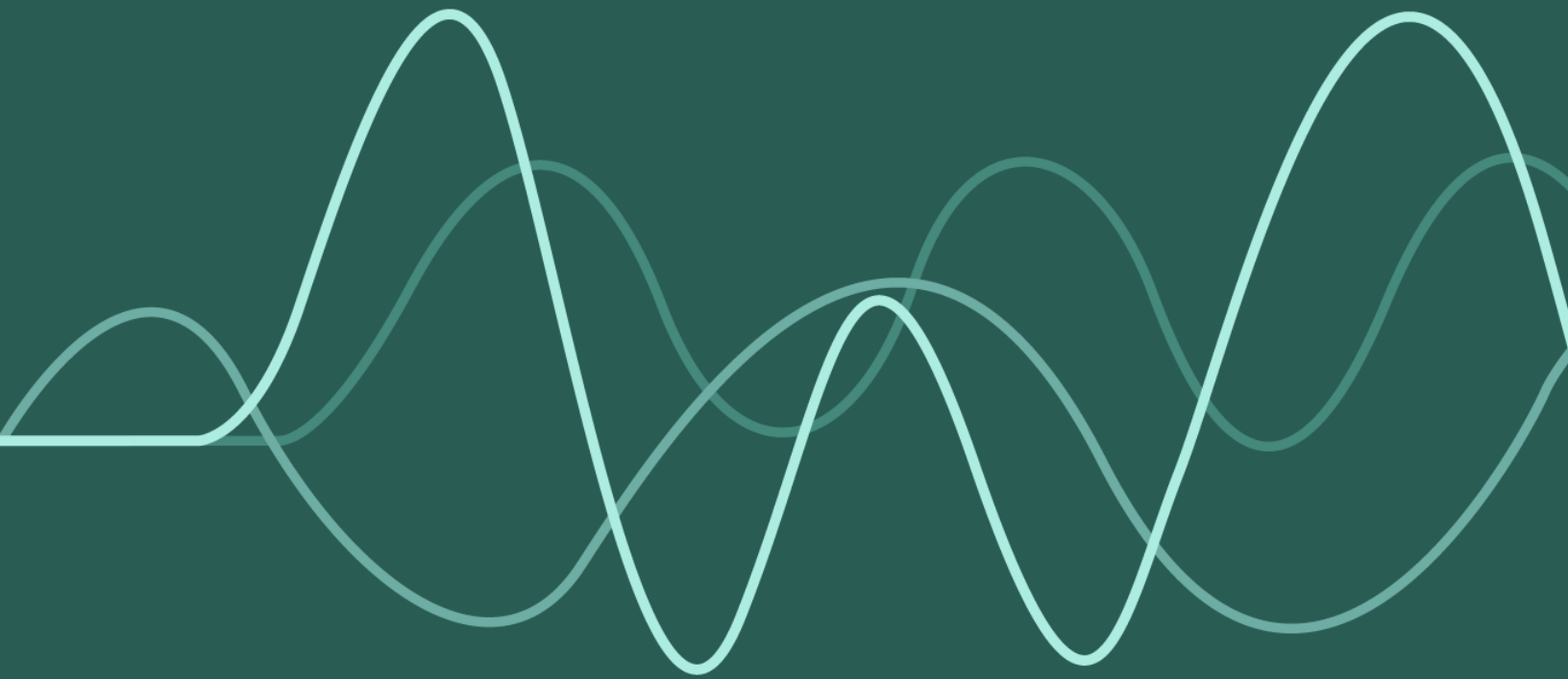


DESIGN OF A DIGITAL SIGNAL FILTERING CIRCUIT



Student:
Zară Antonia-Maria
Group 30433

2021

Content Table

1. Introduction.....	1
1.1. Context.....	1
1.2. Specification.....	1
1.3. Objectives.....	1
2. Bibliographic Study.....	2
3. Analysis.....	6
4. Design.....	9
5. Implementation.....	14
6. Testing and Validation.....	17
7. Conclusions.....	19
8. Bibliography.....	20

1. Introduction

1.1. Context

The purpose of this project is to design, implement and test different digital filters on an FPGA board. The project will have different filters available to use, such as linear, causal, memoryless or recursive filters, which the user will choose, and an input sequency which will represent input signal samples, each at a specific time. The project will output a new signal characterized by a computed modification of the input. The project may be of use to those who wish to an FPGA board for modifying signals, reducing noise or transforming samples for powerful calculations such as state equations in system's theory or in the audio producing field.

1.2. Specifications

The project will be implemented on a Basys3 FPGA board with Xilinx Artix-7 architecture. From its features, the ones relevant to this project are the 4 digit 7 segment display, the pushbuttons, the LEDs and switches, the serial flash, the internal clock and the components for communication with the host (the computer). The samples from the output signals will be displayed on the 4 digit display, while the change in filter will happen on button actions.

The code needed for the implementation will be written in VHDL, using the Vivado Design Suite. This IDE will help us in the process of code programming, design, simulation and testing.

1.3. Objectives

The objective of this project is to have a fully functional routine that will let an user input a sequence of values representing signals behaviour in time and choose a filter from

those available (specified in Design chapter) and get the output result both as a displayed value on the FPGA board's seven segment displays and on the console from Vivado IDE.

2. Bibliographic Study

2.1. Introduction to digital filters

A filter is a medium through which an input signal passes that produces a new signal which in turn represents a modified version of the input. Real life examples of filters are speakers, that might modify the songs that we listen to and their properties, some of us enjoying more bass while the romantics prefer more treble, our vocal chords, music amplifiers or equalizers or a room with its acoustics.

From an engineering point of view, filters are used to remove unwanted bandwidths from a signal^[4], such as random noise, or to extract certain components of a signal^[3]. The following diagram^[3] illustrates the basic idea



A digital filter uses a digital processor to perform numerical calculations on sampled values of the signal. The processor may be a general-purpose computer such as a PC, or a specialised DSP (Digital Signal Processor) chip^[3].

It is important to realize that a digital filter can do anything that a real-world filter can do. That is, all the filters alluded to above can be simulated to an arbitrary degree of precision digitally. Thus, a digital filter is only a formula for going from one digital signal to another. It may exist as an equation on paper, as a small loop in a computer subroutine, or as a handful of integrated circuit chips properly interconnected^[2].

The difference between analog and digital filters is that analog filters are implemented using hardware circuits and technology, while digital filters are defined as a set of computations written as instructions in code. Therefore, digital filters are programmable, thus being easily modifiable in case of need, and much more portable. Another advantage is their complexity, being easier to define complex filters than building them, complexity which is included in their property of stackability.

2.2. Examples of filters

I selected a different computation for each category of digital filters. Considering x_n the input signal and y_n the output of the filter.

1. Linear time invariant (LTI) filters

$$y_n = 2 * x_n + 3 * x_{n-1}$$

Linear filters has the property that its output signal is a linear scale of the input. The time invariant property defines the fact that the signal suffers the same modification regardless of the time of the computation.

2. Time varying filter (**)

$$y_n = x_n + 2 * n * x_{n-1}$$

This filter is a time varying filter because the coefficient of one input sample changes depending on the time of the sample.

3. Causal filters

$$y_n = 4 * x_n + 2 * x_{n-1} + 3 * x_{n-2}$$

This filter is causal because it is a computation only of past and present input samples. To note that this specific filter is also LTI.

4. Recursive digital filters (feedback filters)

$$y_n = 2 * x_n - x_{n-1} + 3 * y_{n-1}$$

Recursive filters, or feedback filters, take into account when computing the output not only past or present states of the input, but the past states of the output signal as well.

5. Nonlinear filters

$$y_n = (x_n)^2 + (x_{n-1})^2$$

Nonlinear filters are defined by a nonlinear function, such as exponential, logarithmic, trigonometric etc.

6. Finite Impulse Response filters (N-Tap example)

$$y_n = h_0 * x_n + h_1 * x_{n-1} + h_2 * x_{n-2}$$

An FIR is a filter has an impulse response of finite period, and as a result of it settles to zero in finite time. "h" here is a function defined on its coefficient.

7. Infinite Impulse Response filters

$$\sum a_k * y_{(n-k)} = \sum b_k * x_{n-k}$$

Infinite impulse response filters are distinguished by having an impulse response which does not become exactly zero past a certain point, but continues indefinitely.

8. Memoryless filters

$$y_n = 3 * (x_n)^3$$

Memoryless filters are filters whose output at time n is not dependent on past inputs or outputs.

These categories are not exclusive, an equation may be a part of multiple categories and has been written under one only as an example, not a strict representation.

3. Analysis

3.1 Use cases

The use cases in this project depend mainly on the validity of the user input, hence recognizing two cases: valid and invalid input options.

The input for our components such as the adder or the multiplier is on 8 bits, therefore the numbers that can be represented are in the range 0-255.

The first use case is for a valid input, state variable being in the range described above. On valid input, the serial communication receiver transforms the bytes information to a series of bits that will be sent to the board. The project takes the input to the currently selected filter which, with the help of adders, multipliers and registers, will compute the output value of the filter. The output is used by the serial communication transmitter and is sent to the console of the PC.

On invalid input, the result will be set to 0, thus an empty filter being used, and perhaps an error led will be lit.

3.2 Components

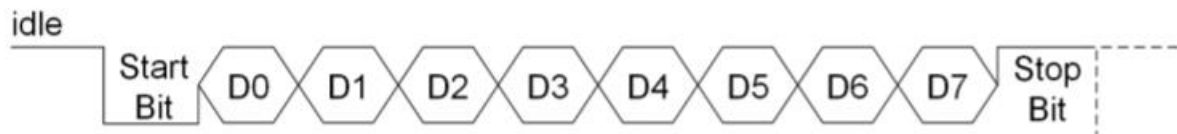
The components that must be analyzed to some degree are the serial communication UART with its receiver and transmitter, the carry lookahead adder and the Wallace tree multiplier that will be used in our design.

3.2.1 Serial Communication – UART

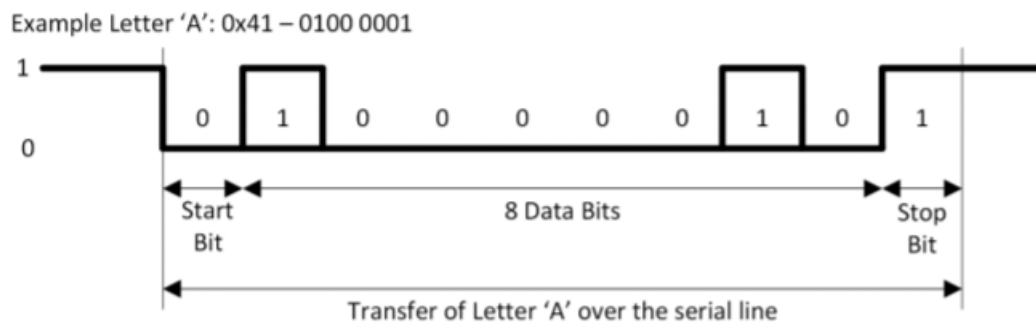
Serial communication is the transmission or reception of data one bit at a time. Today's computers generally address data in bytes or some multiple thereof. A serial port is used to convert each byte to a stream of ones and zeroes as well as to convert streams of ones and

zeroes to bytes. The serial port contains an electronic chip called a Universal Asynchronous Receiver/Transmitter (UART) that actually does the conversion. ^[6]

When transmitting a byte, the UART first sends a START BIT followed by the data (generally 8 bits, but could be 5, 6, 7, or 8 bits), followed by STOP BITS. The sequence is repeated for each byte sent. ^[6]



The start bit is always 0, the data bits are transmitted with the LSB (least significant bit) first and MSB (most significant bit) last and the stop bit is always 1. The data sent through serial communication is encoded using ASCII codes. Assume we want to send the letter 'A' over the serial communication channel. The binary representation of the letter 'A' is 01000001 (0x41hex). ^[6]



The serial communication mechanism will be used in order to give the state variables as input for our filter from console to FPGA, as well as to get the output on console from the computation of the filter result.

3.2.2 Ripple Carry Adder

This type of adders are implemented by several full adders in series, each carry output is connected to the input of the next one. This is a parallel adder and is used for adding n-bit number. It has the advantage of simplicity and low cost, but at the cost of reduced speed. ^[7]

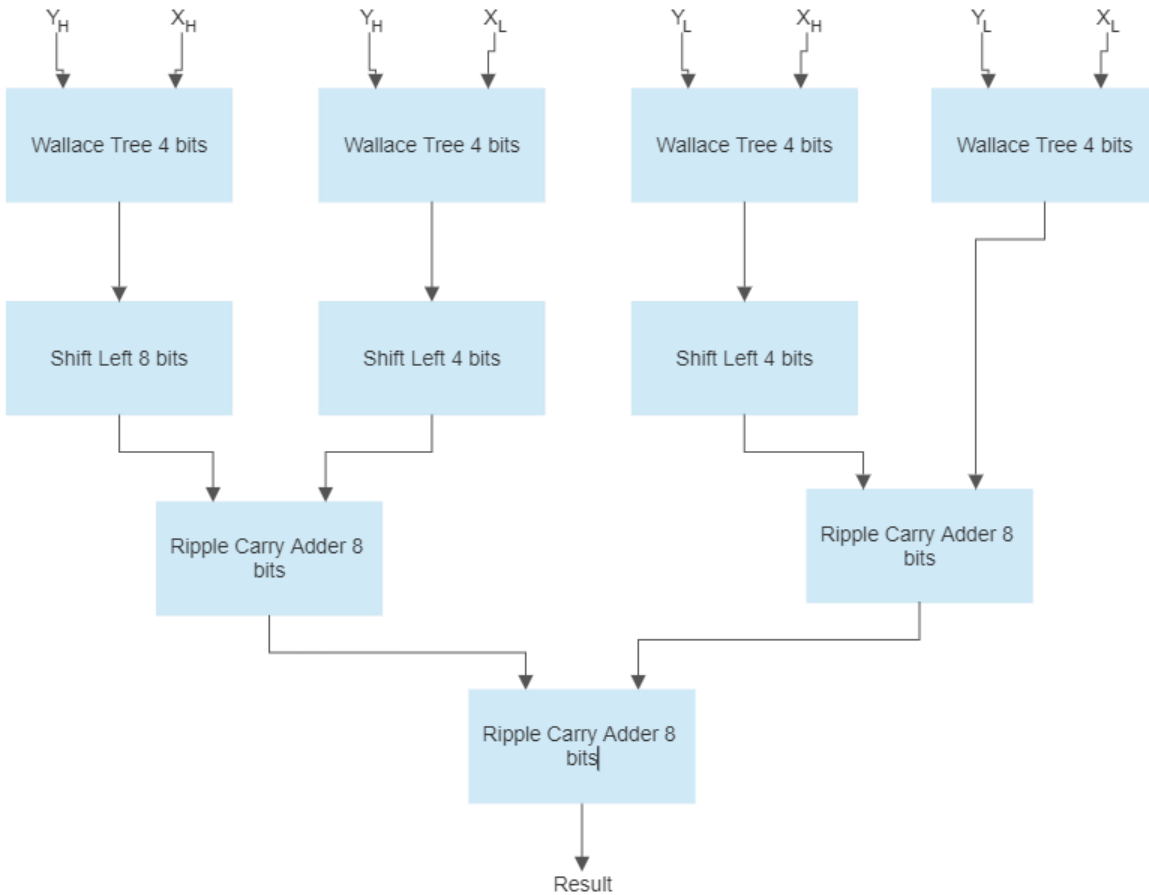
3.2.3 Wallace Tree Multiplier

The Wallace Tree technique is based on combining pairs of partial products with the help of multiple levels of Carry Save Adders. At each level of the tree, the numbers are grouped into three and are added together. The levels continue until only 2 numbers are left to be added and additionally a carry propagate adder is used to add the last 2 numbers and deliver the final result. ^[7]

4. Design

4.1 Design of Components

4.1.1 Multiplier on 8 bits using Wallace Tree

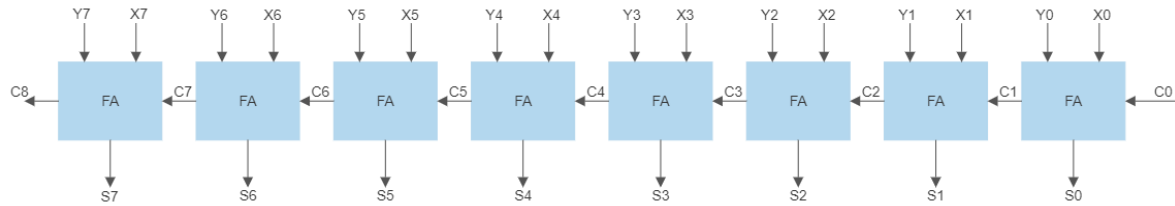


I decided to use numbers represented on 8 bits, therefore I needed a proper multiplier for 8 bit numbers. Considering a Wallace Tree multiplier on 4 bit inputs available, I have divided each operand on the highest and lowest 4 bits and noted the parts accordingly: first operand is divided into X_H and X_L , while the second operand is divided into Y_H and Y_L . The final result is in follows the formula of:

$$Y_H * X_H * 256 + Y_L * X_H * 16 + Y_H * X_L * 16 + Y_L * X_L = Result$$

In order to reduce the complexity of the operation, instead of multiplying with 256 or 16 we shift the intermediate product results by 8 and 4 bits respectively. The 8 bit shift will always result in 0b00000000, which is irrelevant in our case, but is implemented for overall correctness. Note that the Ripple Carry Adders are 8 bit adders.

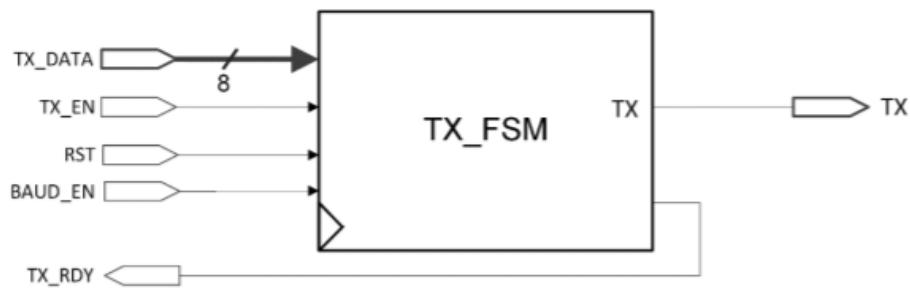
4.1.2 Ripple Carry Adder on 8 bits



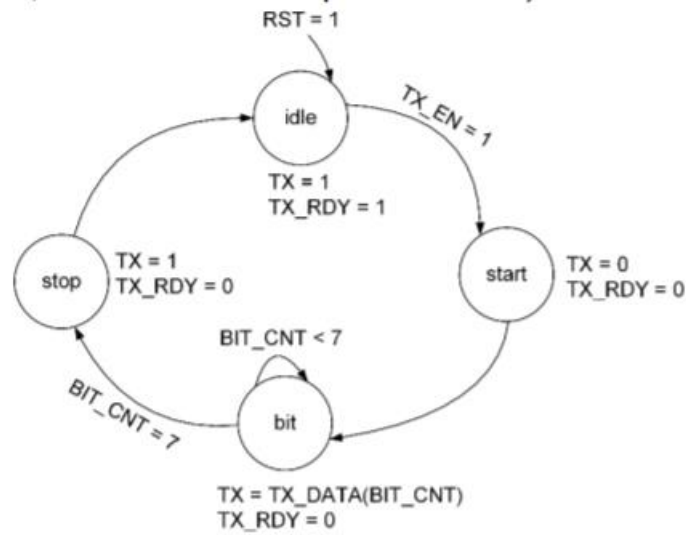
For 8 bit operands we need 8 full adders in order to implement an 8 bit ripple carry adder. For each full adder, the carry is propagated to the next adder and so on until we get the final carry, considered as overflow in our case.

4.1.3 Serial Communication UART

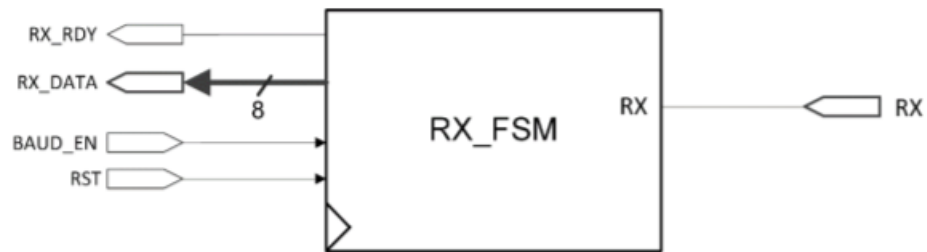
Transmitter



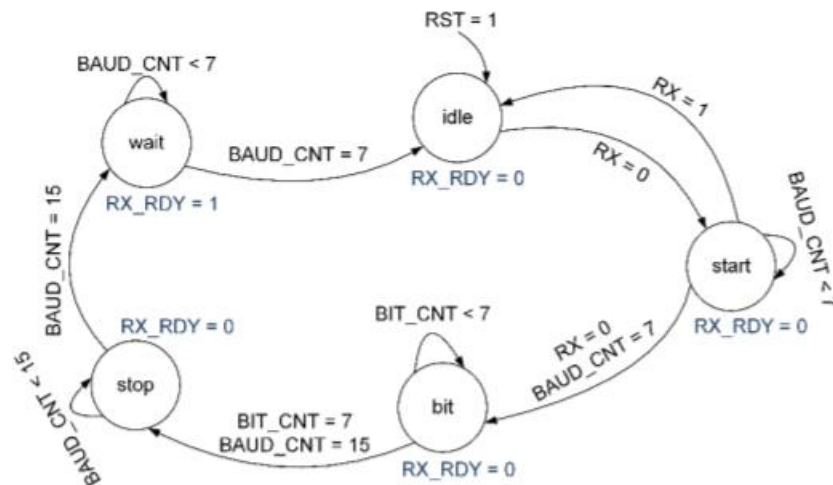
The detailed FSM implementation is presented in the figure below. A state transition is triggered only in the clock cycle when BAUD_Enable is '1'. More details in the source [7].



Receiver

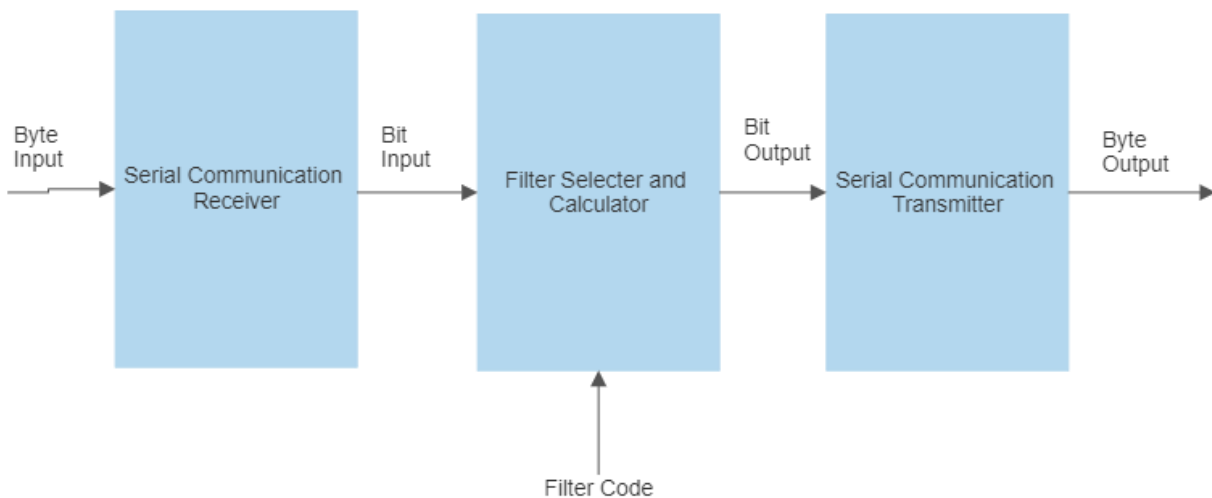


The detailed FSM implementation is presented in the figure below. A state transition is triggered only in the clock cycle when BAUD_ENable is '1'. More details in the source [7].



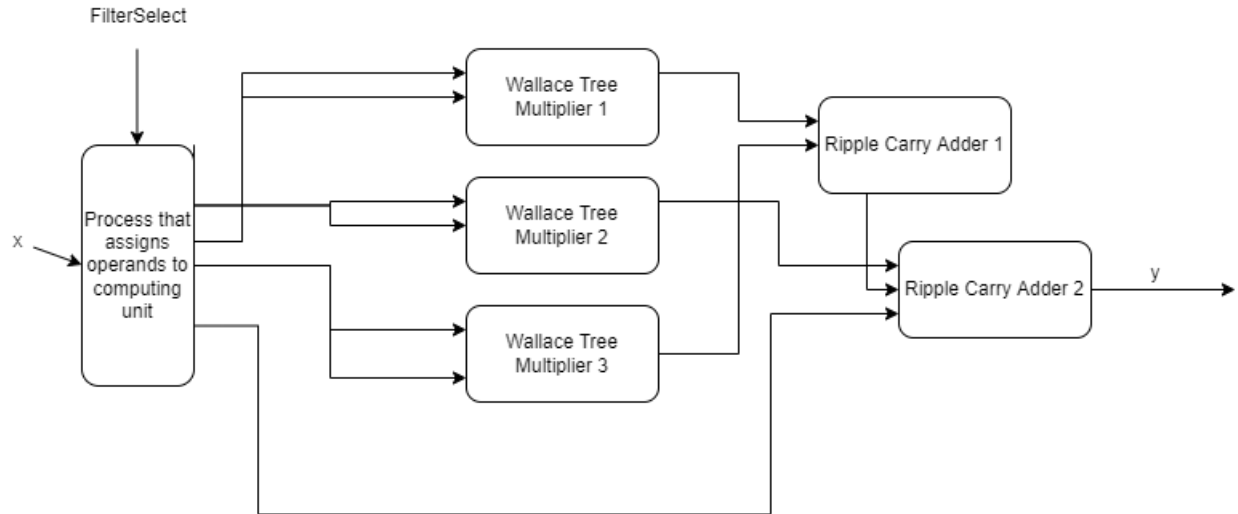
4.2 Project Design

The biggest scheme presents the components regarding the transmission of data to and from the project through the user. This transmission involved both components of the serial



communication UART, presented in the chapters above. The data from the input is evaluated into the filter, whose selection is made from the Filter Code input (thus, knowing which of the 8 filters to use).

The Filter Selector and Calculator will use a decoder in behavioral description in order to pick the wanted filter, while the computations will be made using the multiplier with Wallace Tree components and the Ripple Carry Adder on 8 bits. Any filter uses at most two multipliers and three adders, therefore I have instantiated these exact components. If for any filter, an adder is not used, its operands will pass along another result and add it with zero. For multipliers, if unneeded, it just multiplies the result by one.



5. Implementation

Check my GitHub: <https://github.com/zaraantonia/DigitalSignalFilters>

5.1 Filters

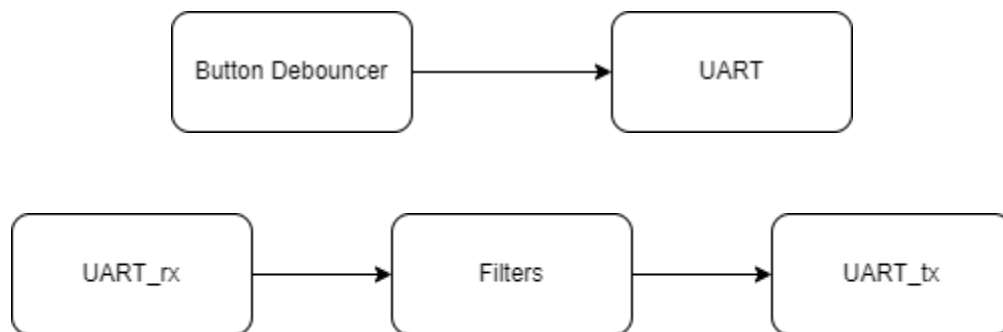
For the filters, the corresponding file contains the port maps of the adders and the multipliers. In order to assign the operands to those unit, I have a process that assigns the operands based on the chosen filter (which comes from the first two switches of the FPGA board). Additionally, there is another process whose only element in the sensitivity list is a signal `new_input` coming from a button of the board. A button press represents the storing and computing of a new input. This process aims to propagate the older inputs to their current value, for example $x(n)$ becoming $x(n-1)$ since a new input has been registered.

The filter unit additionally implements a reset signal, created for easiness of the filter switch.

For some of the filters I had pipelining issues due to my storage of values in signals on rising edge of the clock. I treated each filter individually and solved the problem by adding an extra signal which stores a value that passes it into an “unnatural” stage (for example $x(n-2)$ being passed at the same time with $x(n)$, although it wouldn’t normally happen being sent two stages before).

5.2 UART Communication on FPGA Board

I used bibliography source [8] for the structure of the UART communication between FPGA and the PC, for the files `uart_rx.vhd`, `uart_tx.vhd` and `uart.vhd`. In the UART transceiver, besides the receiver and the transmitter I have additionally instantiated the filters component in order to redirect the input and the output according to the purpose of the project. The baud rate used was chosen arbitrarily as 115200 symbols per second. In the final version of the project, the receiver gets the value from the PC, which uses as input for the filter unit. The output of this unit is then send to the PC through the transceiver. The following scheme represents the elements of “`uart_final.vhd`” and “`uart.vhd`” respectively:



5.3. UART Communication on PC

Because I couldn't find a serial terminal that suited my exact needs in terms of storing and sending data in and to the filtering circuit, I have decided to write my own version in C. The file “`uart_transceiver.cpp`”, according to its name, represents a unit that reads numbers from the users, sends them to the FPGA board and then stores the computed values in a file named “`output.txt`”. I have included instructions and how-to's in the starting lines of the program, in order to aid the user.

```
D:\3.1\ProjectSCSCopy\digital-signal-filter\UART_transceiver_pc\Debug\UART_transceiver_pc.exe

#####
##  DIGITAL SIGNAL FILTERS  ##
#####

Instructions:

    Select one of the following switches from the board:
        00 ->  $2 \cdot x_n + 3 \cdot x_{n1}$ 
        01 ->  $x_n + 2 \cdot n \cdot x_{n1}$ 
        10 ->  $4 \cdot x_n + 2 \cdot x_{n1} + 3 \cdot x_{n2}$ 
        11 ->  $x_n \cdot x_n + x_{n1} \cdot x_{n1}$ 

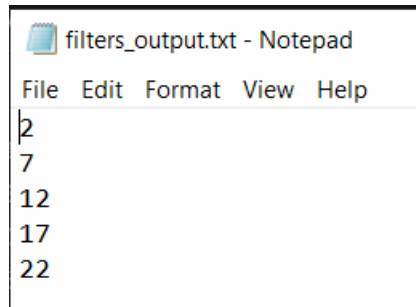
    After every input, press enter and then press the central button on board.
    Every 5 inputs, you will be asked if you want to end.
    For terminating the program, input 'N' when asked.
    After the command for termination, keep pressing enter and the central button
    on the board for result propagation until the program is finished.

Opening serial port...
Serial port opened OK
Start sending numbers? (Y/N):
```

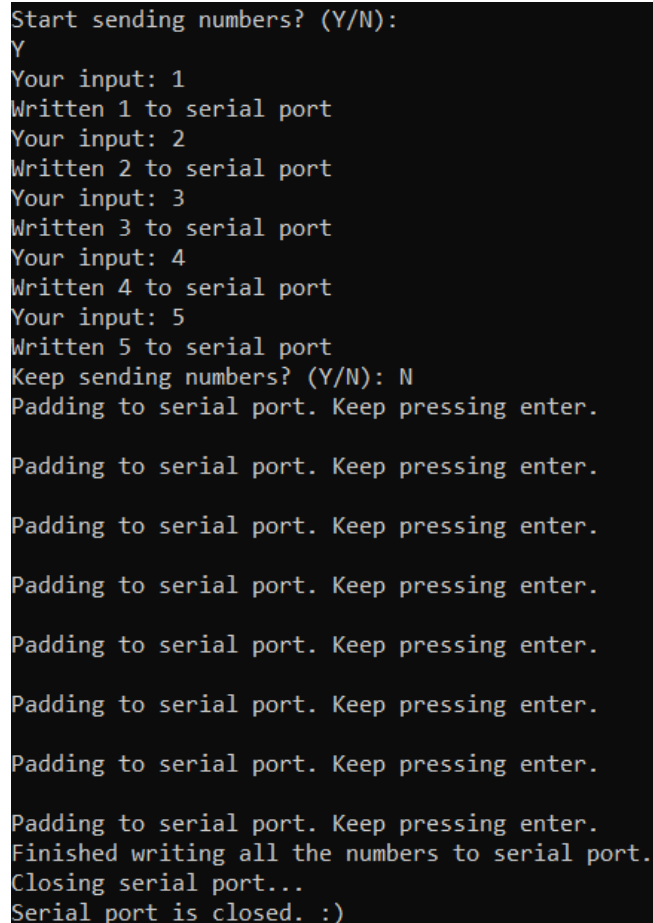
The user is asked if he wants the send numbers. On positive response, the program will wait for inputs as integers. At every five inputs, the user is asked if he wished to continue inputting values. When the data has been sent and a wish for termination has been expressed to the program, the user must press Enter another seven times, which represents the total propagation time of the last inputted value and for the result to be computed. After every input and in the padding stage presented above, the user must additionally press the central button to register the new input on the FPGA board. Do not forget to select the wanted filter and reset the board (using the up button) after each use.

6. Testing and Validation

For testing the filtering circuit on the PC, I have inputted values from 1 to 5 in the terminal, because they were the easiest to compute by hand, and checked the values from the output file with the ones I have calculated by hand, if the values matched that means that the filters are computing correctly.



```
filters_output.txt - Notepad
File Edit Format View Help
2
7
12
17
22
```



```
Start sending numbers? (Y/N):
Y
Your input: 1
Written 1 to serial port
Your input: 2
Written 2 to serial port
Your input: 3
Written 3 to serial port
Your input: 4
Written 4 to serial port
Your input: 5
Written 5 to serial port
Keep sending numbers? (Y/N): N
Padding to serial port. Keep pressing enter.

Padding to serial port. Keep pressing enter.

Padding to serial port. Keep pressing enter.

Padding to serial port. Keep pressing enter.

Padding to serial port. Keep pressing enter.

Padding to serial port. Keep pressing enter.

Padding to serial port. Keep pressing enter.

Finished writing all the numbers to serial port.
Closing serial port...
Serial port is closed. :)
```

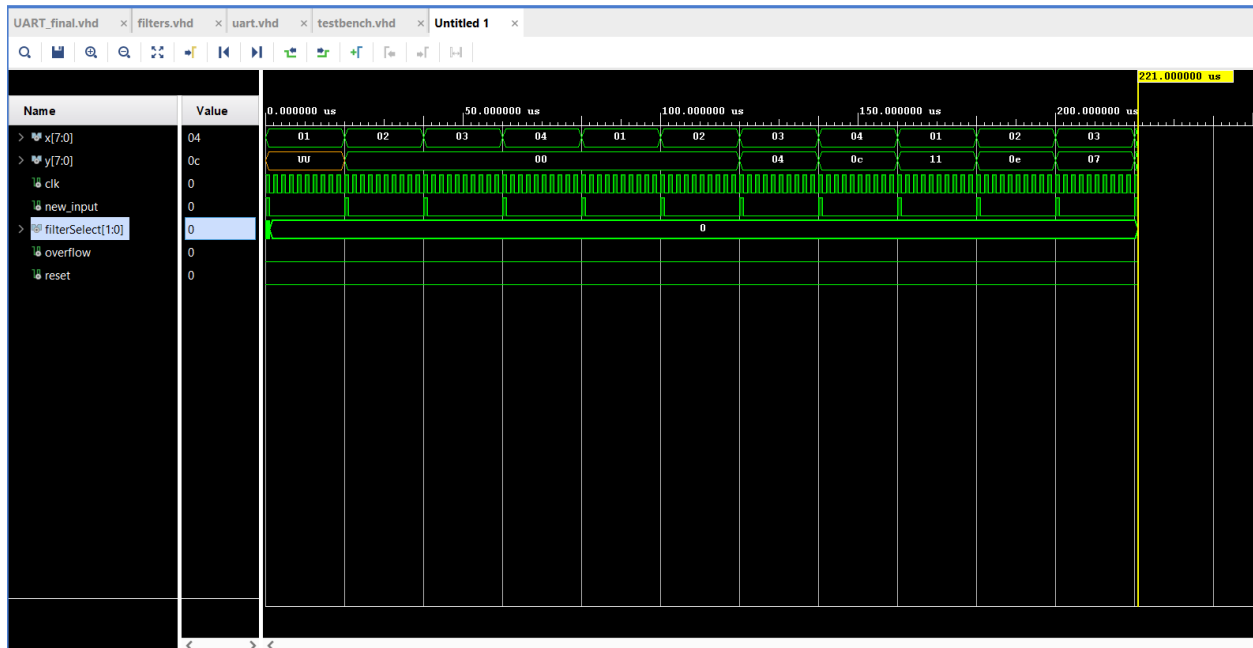
For testing the serial communication between the PC and the FPGA board, I have covered in my code the cases of failed stages in the communication. Therefore, if anything goes wrong, a detailed message will appear on the terminal. Additionally, in order to check the communication on the board's end, I check that the TX associated LED lights up everytime a computed output is sent to the output file, and that the RX associated LED lights up everytime a value has been received from the user through the C UART transceiver.

```
//reading serial port
char byte_to_receive = 0;

if (!ReadFile(hSerial, &byte_to_receive, 1, &bytes_read, NULL)) {
    std::cout << "Reading from port failed. Error: %d.\n", GetLastError();
    CloseHandle(hSerial);
    return (4);
}
else {
    if (index > PROPAGATION_TIME) {
        fprintf(output, "%d\n", (int)byte_to_receive);
    }
}

//writing to serial port
number_to_send = 0;
std::cout << "Your input: ";
scanf("%d", &number_to_send);
if (!WriteFile(hSerial, &number_to_send, 1, &bytes_written, NULL))
{
    std::cout << "Error sending number " << (int)number_to_send << "\n";
    CloseHandle(hSerial);
    return 1;
}
else {
    std::cout << "Written " << (int)number_to_send << " to serial port\n";
    index++;
}
```

In order to test and validate the filtering circuit, I have written a testbench which chooses a filter and inputs some data to the board then I have checked on the simulation if all the signals are acting as expected. If yes, I have considered the filter successful and tested it on the FPGA, where I additionally checked signals and their validity by outputting them on the LEDs.



7. Conclusions

In conclusion, the project has implemented half of the filters proposed in the design stage, four out of eight, but has the extra feature of the C program to input and output data to the filters which has not been featured in the beginning. I have chosen to implement this part, in the detriment of the more complex filters, because I felt that the mixed used of programming languages in a project is a situation that I will probably encounter in the future and because I have an affinity for working with low level C structures and functions. The serial communication part in C was fun to write and research and it felt amazing when it actually worked.

Additionally, regarding the filtering unit, I am aware that I could have chosen an easier to implement multiplier, for example a shift and add multiplier, instead of a personally created 8 bit Wallace Tree out of four 4 bit Wallace Tree multipliers, but the design of the more complex multiplier reminded me of Logic Design and thinking for hours about how data and math should flow in binary. A great project overall. :)

8. Bibliography

1. CS/ECE 5785/6785: Advanced Embedded Software (Fall 2012) [Online]. Available:
<https://my.eng.utah.edu/~cs5785/slides-f08/18-1up.pdf>
2. “Physical Audio Signal Processing For Virtual Musical Instruments and Audio Effects”, Julius O. Smith III, Center for Computer Research in Music and Acoustics (CCRMA), Stanford University [Online]. Available: <https://ccrma.stanford.edu/~jos/pasp/>
3. Physics 123/253, Signals & Noise (Fall 2013), University of California, Davis [Online]. Available: <https://123.physics.ucdavis.edu>
4. Analogic and Digital Circuits course (Fall 2020), Technical University of Cluj-Napoca [Online]. Available: to UTCN students.
5. Signal Processing course at Groningen University [Online]. Available:
<https://www.astro.rug.nl/~vdhulst/SignalProcessing/Hoorcolleges/college07.pdf>
6. Computer Architecture course (Spring 2021), Technical University of Cluj-Napoca [Online]. Available: to UTCN students.
7. Structure of Computer Systems course (Fall 2021), Technical University of Cluj-Napoca [Online]. Available: to UTCN students.
8. UART Interface in VHDL for Basys3 example, Alexey Sudbin [Online]. Available:
<https://www.hackster.io/alexey-sudbin/uart-interface-in-vhdl-for-basys3-board-eef170>