Instituto Tecnológico
y de Estudios Superiores de Occidente

Reconocimiento de validez oficial de estudios de nivel superior según acuerdo secretarial 15018, publicado en el Diario Oficial de la Federación del 29 de noviembre de 1976.

Departamento de Electrónica, Sistemas e Informática

Especialidad en Diseño de Sistemas en Chip

Filtro digital FIR adaptativo y configurable

TRABAJO RECEPCIONAL que para obtener el **DIPLOMA** de
ESPECIALISTA EN DISEÑO DE SISTEMAS EN CHIP

Presenta: **RENÉ SAÚL DÁVILA VELARDE Y RICARDO RAMOS CONTRERAS**

Asesor **DR. JOSÉ LUIS PIZANO ESCALANTE**

Tlaquepaque, Jalisco. Agosto de 2019.

ITESO, Universidad
Jesuita de Guadalajara

Configurable/adaptive digital FIR filter

Specialist Thesis/Project

to achieve the university deegree of

SOC design specialist

Presents:

René Saúl Dávila Velarde - es693213@iteso.mx
Ricardo Ramos Contreras - es693682@iteso.mx

Advisor: José Luis Pizano Escalante, PhD

August 09 2019

Acknowledgments

First of all, we are very grateful to CONACYT for providing us the economic support. Sincere thanks for granting us the opportunity.

To our advisor, José Luis Pizano Escalante for all the time, advice and help thorough the project development. In the same way, we appreciate all the support from Professor Cuauhtémoc Aguilera.

Abstract

In digital audio systems, filters or equalizers are medular modules both in its inputs and outputs. Most of the audio processing is managed with DPSs or FPGAs due to the required mathematical operations (mostly multiplication and addition). The development of an IC of this nature could help on reducing the computational time in a lot of audio systems and getting better filtering results with filter orders up to 64 (which DSP systems can't reach); the computational saved time can be used to add extended and more complex functionality to audio systems without wasting it in a high resolution equalizer.

The initial objective of the adaptive filtering module was to be used as a noise canceller through the hardware implementation of a LMS and NLMS algorithms, this mode was tested with a reference model with both a sine wave test file and a song with induced noise. As an extra functionality predefined filter banks were added (2 LPFs and 2 BPFs) and two I2C-configurable filters for user-defined filtering.

The present document contains the modeling, hardware description (and its respective validation using *ModelSim*), FPGA implementation (and functional verification by using *Signal Tap* and audio quality tests) and all the processes inbetween involved to develop a configurable/adaptive FIR filter into an integrated circuit.

List of acronyms and abbreviations

Table 1: Acronyms

AFM	Adaptive Filter Mode
BPF	Band-Pass Filter
CLK	Clock
CS	Channel Side
DRC	Design Rules Check
DSP	Digital Signal Processor
DUT	Design Under Test
FIR	Finite Impulse Response
FOPT	Filter Option
FPGA	Field-Programmable Gate Array
FPA	Fixed Point Arithmetic
GUI	Graphical User Interface
HDL	Hardware Description Language
IC	Integrated Circuit
IO	Input Output
I2C	Inter Integrated Circuits
I2S	Inter-IC Sound
LEC	Logical Equivalence Checking
LMS	Least Mean Square
LPF	Low-Pass Filter
LR	Left-Right
LSB	Less Significant Bit
MAC	Multiply-Accumulate
MCR	Main Control Register
MSB	Most Significant Bit
MUX	Multiplexer
NLMS	Normalized Least Mean Square
OUTC	Output Configuration
RAM	Random Access Memory
ROM	Read Only Memory
RST	Reset
RTL	Register Transfer Level
SDC	Sensitivity Denominator Constant
SNC	Sensitivity Numerator Constant
UDF	User-Defined Filter
VLSI	Very Large Scale Integration

Contents

List of Tables

List of Figures

Introduction

Integrated circuits (ICs) or chips, changed the electronic industry because it became possible to reduce the size of electronic systems by embedding whole basic circuits into a single package; nowadays a single transistor goes down to few nanometers.

The term "bit" is a common word in digital ciruits and represents the minimum size of data that can be managed by a digital system but the main element is the set of these individual bits, named buses. The drawback of using too many parallel buses is the growth of the project size; most of the time this problem gets reduced by serializing part or all the project, gaining space but losing latency.

The present project intends to cover an indispensable module in audio devices, an equalizer or a set of filters. The system intends to test also the hypothesis *an adaptive filter can be used as noise canceller* by developing the hardware implementation of the well known *least mean square* and *normalized least mean square* algorithms. Given that the core element of these algorithms is a FIR filter, little modifications additions a predefined filter coefficents bank and an user-defined filter to deliver a full audio system that consists of a configurable high resolution equalizer and a noise canceller compilant with I2S bus (left justified) and controlled through an I2C bus.

The document starts with a brief Theoretical Framework [1] that covers basic filter and signals theory, the adaptive filtering algorithms implemented and some information about the two serial buses used in communication and flow, I2C and I2S respectively.

The next section, Theoretical Analysis [2], goes through the general specifications, block diagrams of the system and adaptive filtering hardware implementation, pinout, memory map from the register file, processes scheduling referenced to the I2S bus, the python reference model, the values used in the predefined filters bank and, finally; a block diagram, inputs/outputs and its respective *Modelsim* simulation.

The verilog files were verificated by emulating the system in an FPGA. The FPGA implementation [3] shows the RTL synthesis and the *Signal tap* verification for each module, this result gets compared to the simulation in Theoretical analysis [2]. The Development board software for verification [4] covers the description of the tests used for the functional verification and the result of said tests. The final development chapter is the Silicon implementation [5] shows the result from the VLSI phase. Lastly the Results and conclusions [6] summaries all the gotten results, encountered failures/difficulties and possible upgrades.

Theoretical framework

1.1 Signals

An electrical signal is a function that contains information about a phenomenon. Signals are often separated into analog and digital. Both signals have two representations: time and frequency domain.

1.1.1 Analog signal

Most of the signals encountered in science and/or engineering are analog in nature. These signals are functions continuous in time and amplitude. Which means that can take an infinite number of values in a continuous time domain. Both the input signal and the output signal are in analog form.

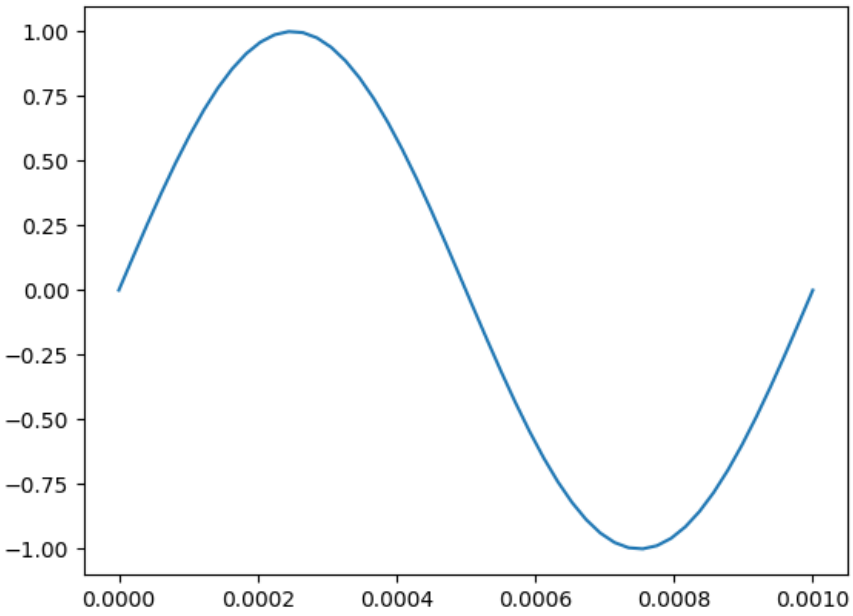

Figure 1.1: Analog signal example

1.1.2 Digital signal

A digital signal is used to mostly as another representation of analog signals. These signals are a sequence of values discrete on time and amplitude, this means that

Figure 1.2: Analog signal processing

there is a finite number of values for the output and it can only change in specified time intervals. Although the digital signal processing diagram shows the digital signal coming from an analog signal, nowadays there're fully digital systems.

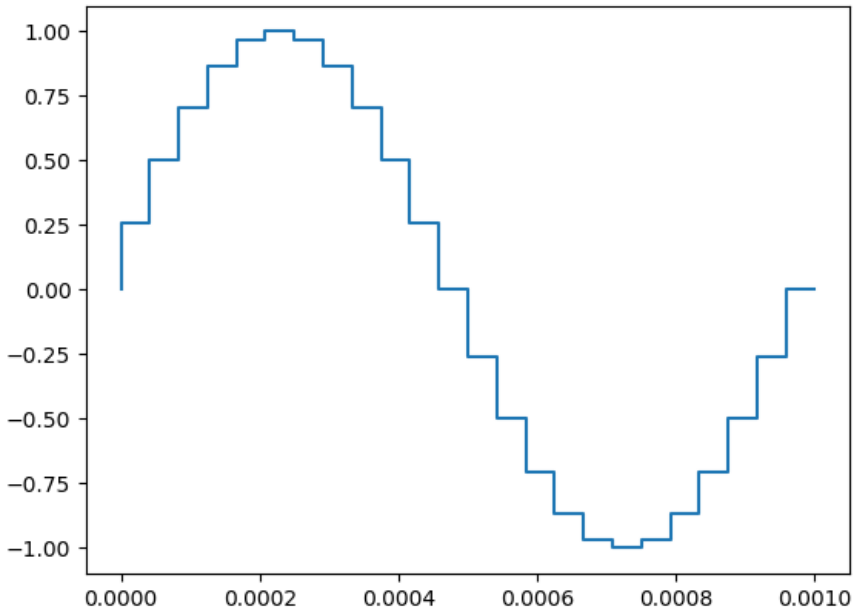

Figure 1.3: Digital signal example

Figure 1.4: Digital signal processing

1.1.3 Time domain

The time domain is where a signal is measured over time. An example is alternating current (AC) signals, which vary its amplitude over a certain time.Figure 1.5 shows the time representation of a 1kHz sine wave.

1.1.4 Frequency domain

The frequency domain is where the amplitude of a signal is measured relative to its frequency. The Laplace transform for continuous-time signals or Z transform

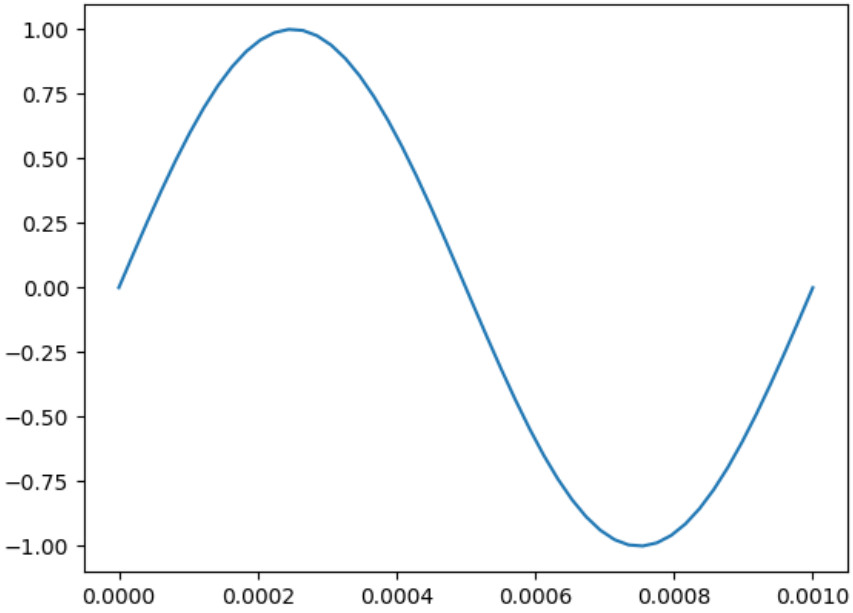

Figure 1.5: 1kHz sine on time domain

for discrete-time signals; give us the spectral information of a signal, it can be also measured using a spectral analyzer. Figure 1.6 shows the frequential representation of a 1kHz sine wave.

1.2 Transfer function

Both analog and digital filters can be considered "black boxes" and characterize the effect on the output at any given input signal. A transfer function is defined as the output divided by the input, both in frequency domain.

$$H(\omega) = \frac{Y(\omega)}{X(\omega)} \tag{1.1}$$

Where:

- $X(\omega)$: input signal in frequency domain.
- $Y(\omega)$: output signal in frequency domain.
- $H(\omega)$: transfer function.

1.3 Filters

An electronic filter is a device that cancels a specific frequency or range of frequencies from a signal. There're two principal topologies: finite impulse response (FIR) and infinite impulse response (IIR). The mathematical difference between both implementations is that the IIR filter uses part or all the output of the filter as input,

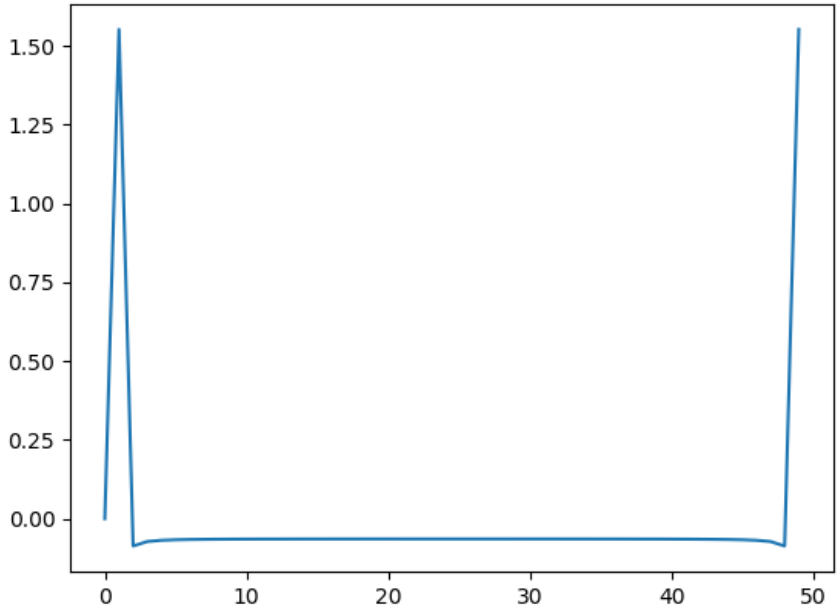

Figure 1.6: 1kHz sine on frequency domain

making it a recursive function. Therefore, a FIR filter has a finite duration, since it only depends on his own values.

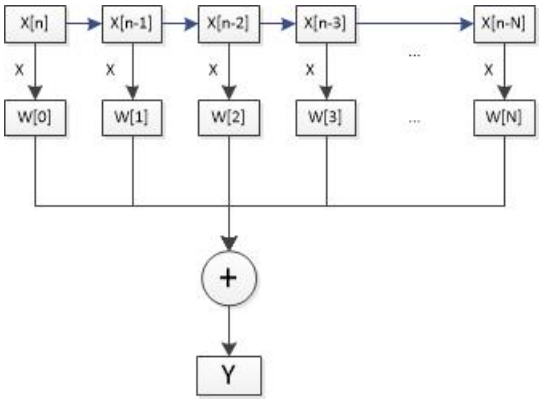

Figure 1.7: General FIR topology

1.3.1 Analog filters

An analog filter is any filter which operates with continuous time signals (analog signals). Analog filters are made of analog components such as resistors, capacitor, inductors, and op amps.

1.3.2 Digital filters

Most digital filters are an approximation from the analog system. This can be achieved by using convolution between a sampled signal and a sampled filter transfer

function. The most common topology is to previously have the time representation of the filter transfer function and convolute it with the time domain signal; more complex systems transforms an input signal to its frequency domain representation (using fast Fourier transform), multiply it by the frequency domain transfer function and later return it to time domain (using inverse fast Fourier transform). As the sample frequency and bit number of the data increases, the digital response approaches the analog one. Unlike analog filters, this type of filters has an upper filter frequency limit due to aliasing effect, to avoid this problem the sampling frequency must be at least 2 times bigger than the highest frequency component of the signal (Nyquist theorem).

1.3.3 Adaptive filters

The adaptive filters (can be found in both analog and digital implementations) are a computational device, the general objective is to model the iterative interaction between two signals. Most of the adaptive filters uses the general FIR topology, but these ones can be modified using a feedback loop. Due to the computational complexity that the mathematical operations require, these filters are mainly implemented in FPGAs or DSPs. Figure 1.8 [1] shows a general topology for every adaptive filter.

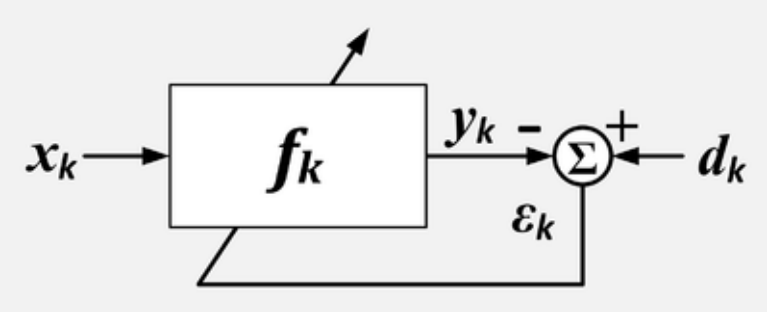

Figure 1.8: General adaptive filter topology

LMS algorithm

LMS stands for "Least mean squares". The objective is to generate an adaptive filter, whose coefficients are continuously modified in a way that the noise from the signal gets eliminated. The filter parameters are:

- $x(n)$: signal to be filtered.
- N: filter order.
- $w(n)$: actual filter coefficients.
- $y(n)$: filtered signal, defined as $x(n) * w(n)$[2].
- $d(n)$: desired signal.
- $e(n)$: filtered signal error, defined as $d(n) - y(n)$.

[1] Obtained from: https://en.wikipedia.org/wiki/Adaptive_filter#/media/File:Adaptive_Filter_Compact.png
[2] The '*' symbol refers to convolution

- μ: step size, defines the convergence rate of the filter.

With a stochastic gradient descent method, which estimates the error at the current time, the new filter values can be calculated with each iteration as:

$$w(n+1) = w(n) + \mu x(n) e(n) \tag{1.2}$$

NLMS algorithm

NLMS stands for "Normalized least mean squares". This algorithm solves a convergence problem of the traditional LMS, normalizing the input signal and reducing the sensibility factor of the input signal power.

$$w(n+1) = w(n) + \frac{\mu x(n) e(n)}{||x(n)||^2} \tag{1.3}$$

1.4 I2S bus

Stands for Inter-IC sound, consists of an electrical serial bus interface standard used mainly for connecting (and establish a data communication) digital devices together. The bus consists of at least three lines (shown in figure 1.9 [3]):

- I2S_CLK: bit clock line.
- I2S_LR: determines if the current word is the left or right channel (both read and write).
- I2S_ADC/I2S_DAC: the serial input or output audio data.

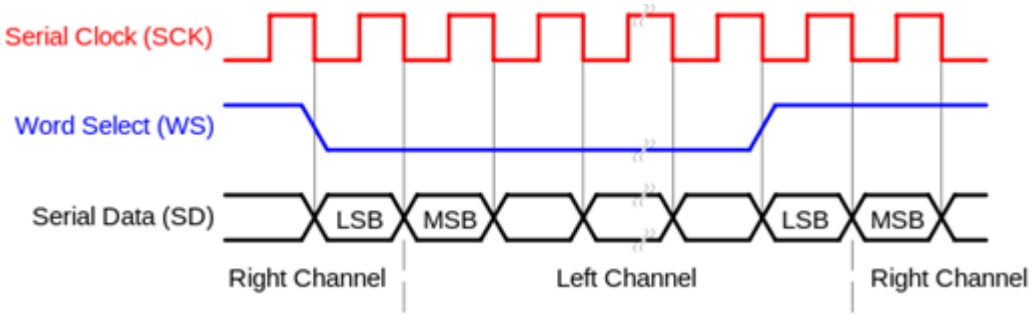

Figure 1.9: I2S timing

1.5 I2C bus

Stands for Inter-Integrated Circuit, it's a synchronous, multi-master, multi-slave, single-ended electrical serial bus for connecting devices together. The I2C data frame is shown in figure 1.10 [4]. The bus consists of two signals:

- SCL: serial clock.

[3] Obtained from: https://commons.wikimedia.org/wiki/File:I2S_Timing.svg
[4] Obtained from: https://cdn.sparkfun.com/assets/6/4/7/1/e/51ae0000ce395f645d000000.png

- SDA: serial data.

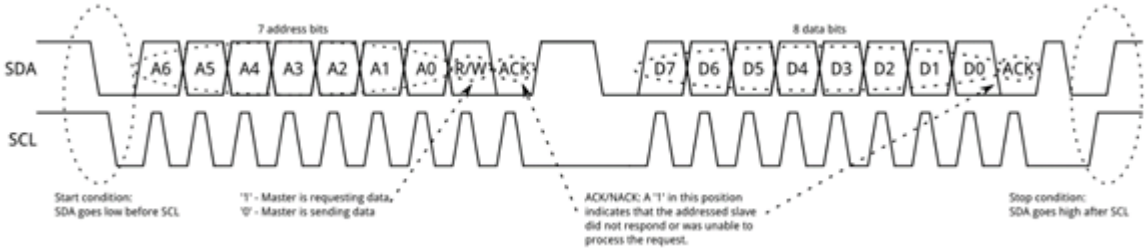

Figure 1.10: I2C timing

Theoretical analysis

The project consists of an audio system that should:

- Read audio data from I2S bus.
- Filter using the I2S bus signals only.
- Write the filtered or bypassed signal from I2S bus.
- Adapt and cancel external noise from the input signal by using either LMS or NLMS algorithm.
- Have 4 pre-defined filters (2 low-pass and 2 bypass) and 2 writable filters, converting the system also as a regular FIR filter.
- - Have an I2C, capable of:
  - Writing the filter coefficients.
  - Writing the control register.
- Internally add and multiply using fixed point operations.

2.6 General diagram

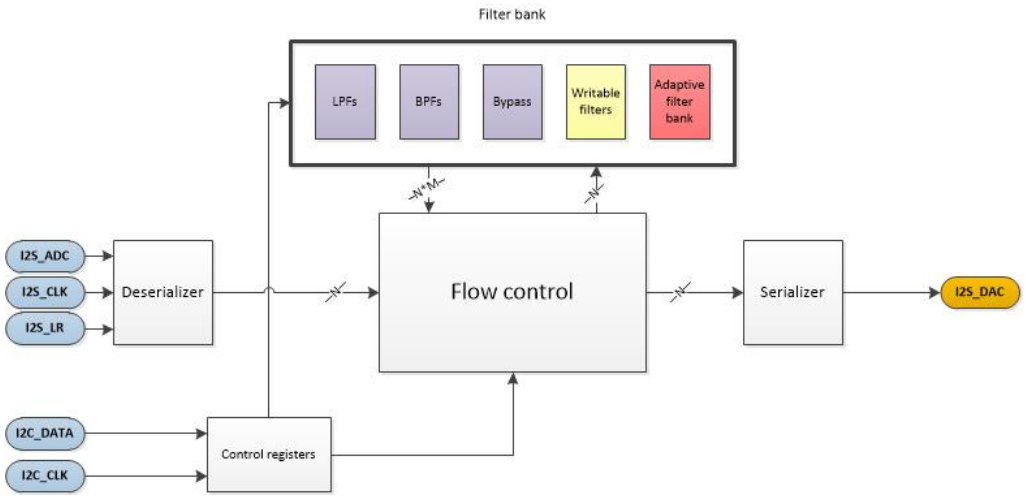

Figure 2.11: General configurable/adaptive FIR filter diagram

The figure 2.12 shows a basic diagram (represents the "Flow control" module) of a single channel filtering; changing the filter coefficients used will result in a

different response (LPF, HPF, etc). The stereo implementation of the filter will only be needed to add a second module with the negative edge of the I2S_LR.

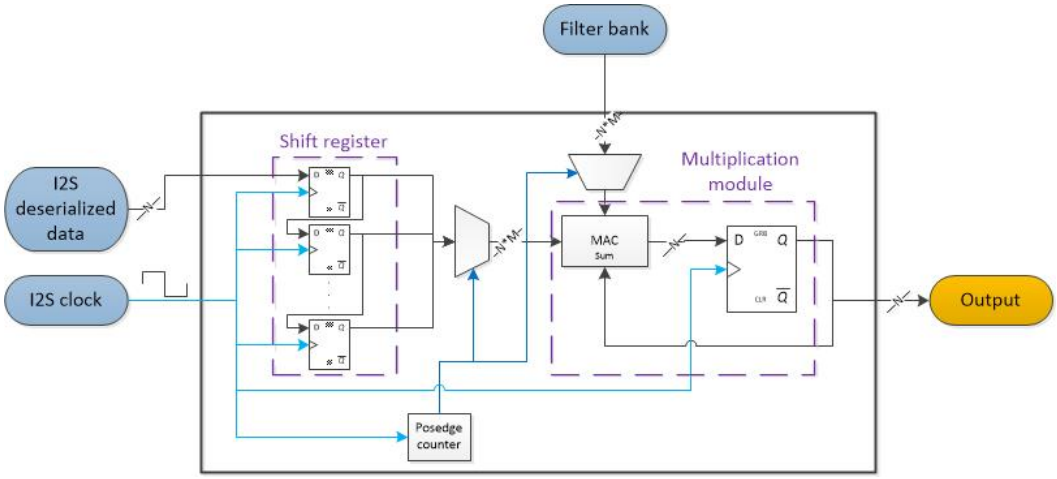

Figure 2.12: Basic mono filtering diagram

2.7 Adaptive filtering implementation

The theoretical coefficient update algorithms used in the design are described in equations 1.2 and 1.3. The LMS filtering method is exactly the same as defined in equation 1.2. For the NLMS implementation an additional constant (β) will be added. The signal power ($||x(n)||^2$) will, in certain time periods, be zero; therefore the added β constant will prevent divisions by zero.

$$w(n+1) = w(n) + \frac{\mu x(n) e(n)}{\beta + ||x(n)||^2} \tag{2.4}$$

2.8 Pinout

Figure 2.13 describes the black box diagram from the below pins.

- **Inputs:**
  - I2C_SCL: I2C bus clock.
  - I2C_SDA: I2C bus serial data.
  - I2S_CLK: I2S bus clock.
  - I2S_ADC: I2S bus serial data from the ADC.
  - I2S_LR: I2S bus data left/right selector.
  - rst: external signal to set initial conditions.
- **Inouts:**
  - I2C_SDA: I2C bus serial data.
- **Outputs:**

– I2S_DAC: I2S bus serial data to the DAC.

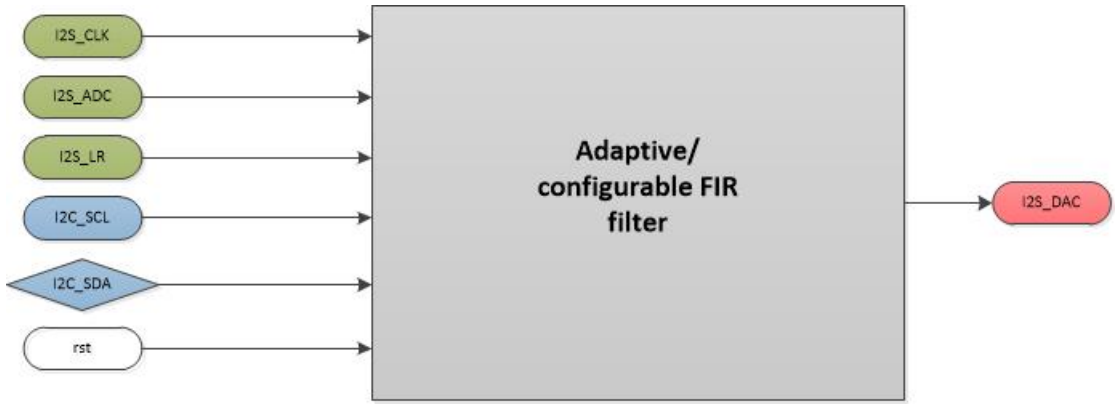

Figure 2.13: Black box diagram

2.9 Memory map

Table 2.2 shows the complete memory map of the design. The acronym definitions can be found in table 2.3.

Table 2.2: Memory map

Address	**B7**	**B6**	**B5**	**B4**	**B3**	**B2**	**B1**	**B0**	**NAME**
0h0	NU	CS	AFM	FOPT			OUTC		MCR
0h1	NU	I2C_SLAVE							I2C
0h2	LOW								SNCL
0h3	HIGH								SNCH
0h4	LOW								SDCL
0h5	HIGH								SDCH
0h6	LOW								UDF0_L00
0h7	HIGH								UDF0_H00
0h8	LOW								UDF0_L01
0h9	HIGH								UDF0_H01
...	...								UDF0_L/Hnn
0h24	LOW								UDF0_L15
0h25	HIGH								UDF0_H15
0h26	LOW								UDF1_L00
0h27	HIGH								UDF0_H00
0h28	LOW								UDF1_L01
0h29	HIGH								UDF1_H01
...	...								UDF1_L/Hnn
0h44	LOW								UDF1_L15
0h45	HIGH								UDF1_H15

Table 2.3: Register file naming

Name	Full name
MCR	Main Control Reigster
I2C	I2C Slave Address
SNCL	Sensitivity Numerator Constant, Low Part
SNCH	Sensitivity Numerator Constant, High Part
SDCL	Sensitivity Denominator Constant, Low Part
SDCH	Sensitivity Denominator Constant, High Part
UDF0_L00	User-Defined Filter 0, Low part 1st coefficient
UDF0_H00	User-Defined Filter 0, High part 1st coefficient
UDF0_L01	User-Defined Filter 0, Low part 2nd coefficient
UDF0_H01	User-Defined Filter 0, High part 2nd coefficient
UDF0_Hnn	User-Defined Filter 0 intermmediate bits
UDF0_L15	User-Defined Filter 0, Low part last coefficient
UDF0_H15	User-Defined Filter 0, High part last coefficient
UDF1_L00	User-Defined Filter 1, Low part 1st coefficient
UDF1_H00	User-Defined Filter 1, High part 1st coefficient
UDF1_L01	User-Defined Filter 1, Low part 2nd coefficient
UDF1_H01	User-Defined Filter 1, High part 2nd coefficient
UDF1_Hnn	User-Defined Filter 1 intermmediate bits
UDF1_L15	User-Defined Filter 1, Low part last coefficient
UDF1_H15	User-Defined Filter 1, High part last coefficient

2.9.1 Main Control Register (MCR)

This register (figure 2.4) contains the main output configuration options.

Table 2.4: Main control register

Address	B7	B6	B5	B4	B3	B2	B1	B0
0h0	NU	CS	AFM	FOPT			OUTC	

Output configuration (OUTC)

Controls the main output multiplexer. Described in table 2.5. **Set to 0x1 by default (bypass).**

Table 2.5: OUTC configuration

Binary	Hex	Configuration	Details
00	0h0	No output	The output is set to 0 at anytime
01	0h1	Bypass	The input data is connected directly to the output
10	0h2	Filter output	Outputs the filter result
11	0h3	Not used	Set to 0

Filtering options (FOPT)

If the OUTC register is configured as "Filter output" this register will define the filter type that will be driven to the output. Described in table 2.6. **Set to 0x0 by default (LFP1).**

Table 2.6: FOPT configuration

Binary	Hex	Configuration	Details
000	0h0	LPF0	Low pass filter, calculated for 500 Hz, the minimum cutoff frequency for WL = 16 bits and N = 16 coefficients is 1894.92 Hz
001	0h1	LPF1	Low pass filter, calculated for 5 kHz
010	0h2	BPF0	Band pass filter calculated for 1 kHz - 20 kHz
011	0h3	BPF1	Band pass filter calculated for 4 kHz - 10 kHz
100	0h4	UDF0	First I2C user-writable filter
101	0h5	UDF1	Second I2C user-writable filter
110	0h6	AF	Adaptive filtering function
111	0h7	NU	Not used

Adaptive filter mode (AFM)

If the OUTC register is configured as "Filter output" and the FOPT is configured as "AF", this register will define from either LMS or NLMS algorithm for noise cancelling. Described in table 2.7. **Set to 0x0 by default (LMS).**

Table 2.7: AFM configuration

Binary	Hex	Configuration	Details
00	0h0	LMS	Least mean square algorithm
01	0h1	NLMS	Normalized least mean square algorithm

Channel side (CS)

Selects/swaps the I2S_ADC left and right channels. This module is specially used for the adaptive filter mode. By default, the noise signal is set to the left channel but if the noise signal comes in the right channel, this register can be set to 0x1. When used in other filter types this register will invert left and right channels in the output. Described in table 2.8. **Set to 0x0 by default (LL).**

Table 2.8: CS configuration

Binary	Hex	Configuration	Details
00	0h0	LL	Set left channel as default I2S left channel
01	0h1	LR	Swap left and right channels

2.9.2 I2C slave address (I2C)

This register (table 2.9) is a 7-bit data, represents the slave address for the I2C module. **Set to 0x45 by default.**

Table 2.9: I2C slave address register

Address	B7	B6	B5	B4	B3	B2	B1	B0
0h1	NU	I2C SLAVE						

2.9.3 Adaptive filtering numerator constant (SNCH/L)

This register (table 2.10) sets the LMS and NLMS sensitivity numerator constant (μ constant) for equations 1.2 and 2.4. **Set to 0x3FFF by default (0.499969482421875 in fixed point format).**

Table 2.10: Sensitivity numerator constant register

Address	B7	B6	B5	B4	B3	B2	B1	B0
0h2	SNC LOW							
0h3	SNC HIGH							

2.9.4 Adaptive filtering denominator constant (SDCH/L)

This register (table 2.11) sets the NLMS sensitivity numerator constant (β constant) for equation 2.4. This constant is ignored when LMS algorithm is selected. **Set to 0x4000 by default (0.5 in fixed point format).**

Table 2.11: Sensitivity denominator constant register

Address	B7	B6	B5	B4	B3	B2	B1	B0
0h4	SDC LOW							
0h5	SDC HIGH							

2.9.5 User defined filter 0 (UDF0)

This register (table 2.12) is the first writable filter of the system. **Set to 0x0 by default.**

Table 2.12: User defined filter 0 register

Address	B7	B6	B5	B4	B3	B2	B1	B0
0h6	UDF0 LOW 0							
0h7	UDF0 HIGH 0							
0h8	UDF0 LOW 1							
0h9	UDF0 HIGH 1							
...	UDF0 ...							
0h24	UDF0 LOW 15							
0h25	UDF0 HIGH 15							

2.9.6 User defined filter 1 (UDF1)

This register (table 2.12) is the second writable filter of the system. **Set to 0x0 by default.**

Table 2.13: User defined filter 1 register

Address	B7	B6	B5	B4	B3	B2	B1	B0
0h26	UDF0 LOW 0							
0h27	UDF0 HIGH 0							
0h28	UDF0 LOW 1							
0h29	UDF0 HIGH 1							
...	UDF0 ...							
0h44	UDF0 LOW 15							
0h45	UDF0 HIGH 15							

2.10 Processes timing

Figures 2.14 and 2.15 show the regular/adaptive filtering processes timing respectively [5]. The top signals can be found in both Regular FIR filtering and Adaptive filtering; *I2S_CLK* and *I2S_LR* are the main control signals, where:

- *I2S_CLK*: general clock signal.
- *I2S_LR*: left-right synchronization signal, if high, the left channel data is provided by the ADC and the output DAC taken by the DAC; whilst low state selects the right channel.

The subsequent highlighted values represent the processes needed for both Regular FIR and Adaptive filtering respectively. The nomenclature used in the diagrams to differentiate left from right channel processing over time is given by: [**channel**][**number of data**].

- channel: can be either L (for left channel) or R (for right channel).

[5] For ease, data numbers where separated by colors.

- number of data: starts at zero, representing the beggining of the data frame and increments per each positive edge of the *I2S_LR*, which indicates the start of the next data for both channels.

For example, if a L3 is in the Deserialize row, it means that the third left data will be processed by the Deserialize module for 16 *I2S_CLK* cycles or half a cycle from *I2S_LR* in the time given by the column intersection.

Regular FIR filtering

I2S_CLK	xN	xN	xN	xN	xN	xN	xN	xN	xN	xN	xN	xN	xN
I2S_LR													
DESERIALIZE	L0	R0	L1	R1	L2	R2	L3	R3	L4	R4			
PUSH		L0	R0	L1	R1	L2	R2	L3	R3	L4	R4		
MULTIPLY			L0	R0	L1	R1	L2	R2	L3	R3	L4	R4	
SERIALIZE				L0	R0	L1	R1	L2	R2	L3	R3	L4	R4

Figure 2.14: Regular filtering process timing

Adaptive filtering

I2S_CLK	xN	xN	xN	xN	xN	xN	xN	xN	xN	xN	xN	xN	xN
I2S_LR													
DESERIALIZE	L0	R0	L1	R1	L2	R2	L3	R3	L4	R4			
PUSH		L0	R0	L1	R1	L2	R2	L3	R3	L4	R4		
MULTIPLY			L0	R0	L1	R1	L2	R2	L3	R3	L4	R4	
SAVE BUS			L0		L1		L2		L3		L4		
POWER			L0		L1		L2		L3		L4		
WRITE NEW COEFFICIENT				L0		L1		L2		L3		L4	
REPEAT LEFT CHANNEL				R0		R1		R2		R3		R4	
SERIALIZE				L0	R0	L1	R1	L2	R2	L3	R3	L4	R4

Figure 2.15: Adaptive filtering process timing

2.10.1 Processes definition

- Deserialize: I2S data comes from the DAC serialized, this process converts it to a parallel bus of 16 bits.
- Push: once there's a parallel data available, it is entered to the shift register, shifting all previously entered data to the right and overflowing the oldest data.
- Multiply: perform the 16 words calculation given by: $y(n) = \sum_{n=0}^{15} w(n)x(n)$.
- Serialize: serializes the output parallel data to be written to the DAC.
- Save bus: as seen both figures 2.14 and 2.15 it takes two full cycles of *I2S_LR*, therefore the shift register needs to be saved before the next data gets pushed in and the subsequent calculations are congruent with the actual processed data.
- Power: calculate the actual quadratic power of the shift register: $||x(n)^2|| = \sum_{n=0}^{15} x(n)x(n)$, to be used in the adaptive filter (only used in the Adaptive filtering scheduling).
- Write new coefficient: calculate and update the next coefficient given by equation 2.4 (only used in the Adaptive filtering scheduling).

- Repeat left channel: when using Regular FIR filtering, right and left channels gets filtered by using the same coefficients, but when using Adaptive filtering, the left channel (data and noise) will be filtered based on the right channel (noise), this creates a mono output signal; this module delays the left channel output 16 clocks to the right channel, replicating the data and converting it once more to a stereo signal (only used in the Adaptive filtering scheduling).

2.11 Reference model

The reference model was implemented in Python, the test signal are formed from the following signals:

- Base frequency: 1.76 kHz.
- First noise frequency: 440 Hz.
- Second noise frequency: 15 kHz.

The functionality can be proven by filtering all but the 4th harmonic of the signal, this means that the output signal will eliminate the equivalent of the base frequency of a square signal and what could be high frequency noise.

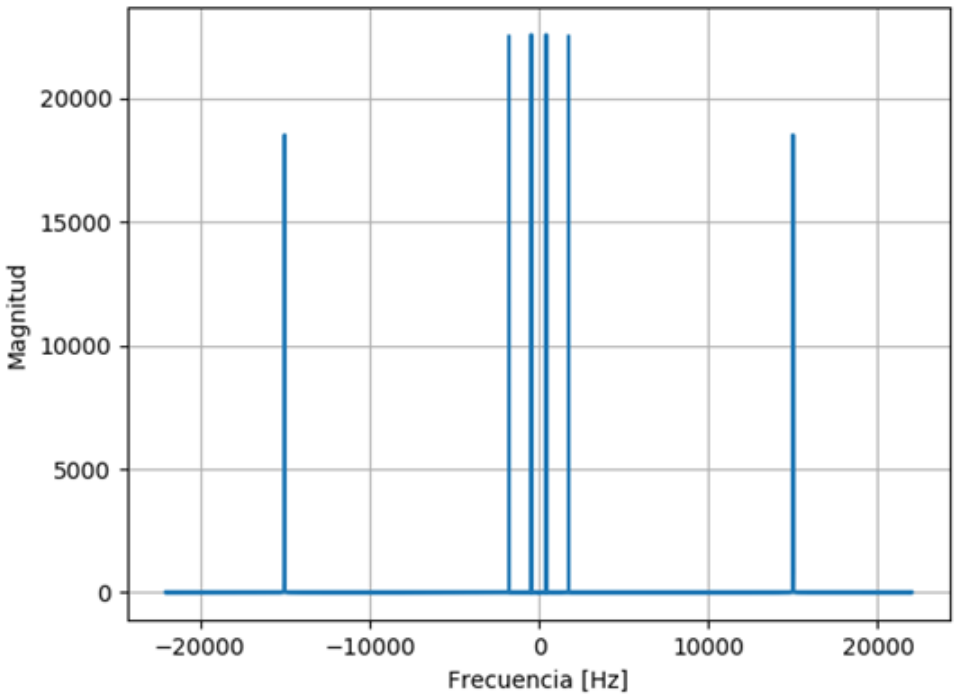

Figure 2.16: Signal with noise in frequency domain

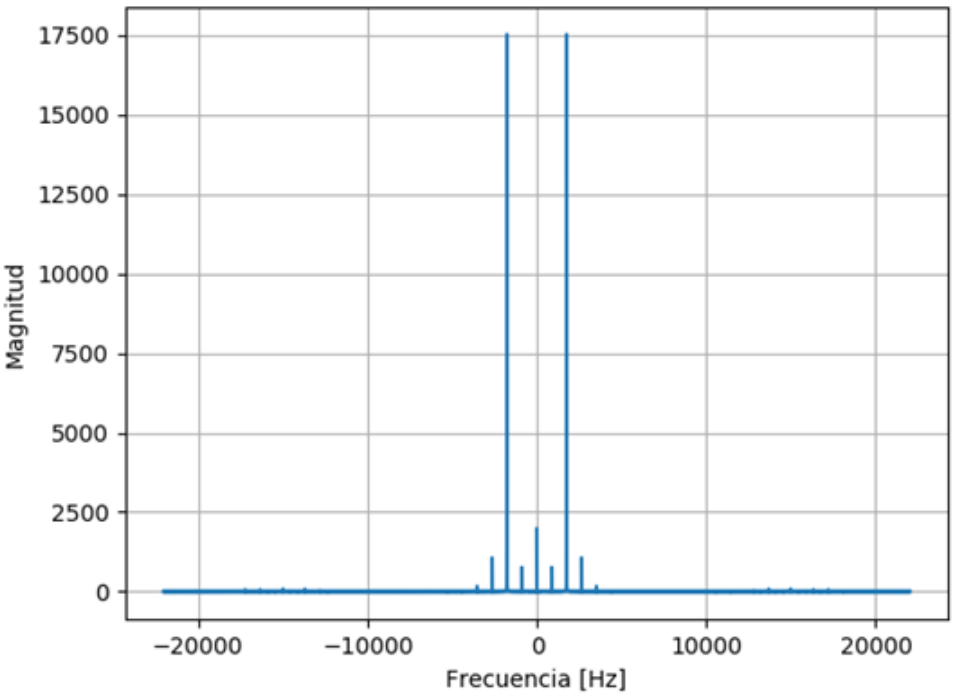

Figure 2.17: Filtered signal in frequency domain

2.12 Filter bank values

The filter values of LPF0, LPF1, BPF0 and BPF1 are shown in tables 2.14, 2.15, 2.16 and 2.17 respectively; the time and frequency representations of the filter values are shown in figures 2.18, 2.19, 2.20 and 2.21 respectively.

Table 2.14: LPF0 values

Coefficient number	Continuous value	Fixed-point representation
0	0.009368896484375	0b0000000100110011
1	0.01422119140625	0b0000000111010010
2	0.027862548828125	0b0000001110010001
3	0.048187255859375	0b0000011000101011
4	0.07171630859375	0b0000100100101110
5	0.094390869140625	0b0000110000010101
6	0.11218261718755	0b0000111001011100
7	0.1219482421875	0b0000111110011100
8	0.1219482421875	0b0000111110011100
9	0.1121826171875	0b0000111001011100
10	0.094390869140625	0b0000110000010101
11	0.07171630859375	0b0000100100101110
12	0.048187255859375	0b0000011000101011
13	0.027862548828125	0b0000001110010001
14	0.01422119140625	0b0000000111010010
15	0.009368896484375	0b0000000100110011
Cutoff frequency		1894.9218749999998 Hz

Table 2.15: LPF1 values

Coefficient number	Continuous value	Fixed-point representation
0	0.003265380859375	0b00000001101011
1	0.006988525390625	0b00000011100101
2	0.01123046875	0b00000101110000
3	0.002136230468755	0b00000001000110
4	0.038604736328125	0b0000010011110001
5	0.1146240234375	0b0000111010101100
6	0.20281982421875	0b0001100111110110
7	0.262725830078125	0b0010000110100001
8	0.262725830078125	0b0010000110100001
9	0.20281982421875	0b0001100111110110
10	0.1146240234375	0b0000111010101100
11	0.038604736328125	0b0000010011110001
12	0.00213623046875	0b00000001000110
13	0.01123046875	0b00000101110000
14	0.006988525390625	0b00000011100101
15	0.003265380859375	0b00000001101011
Cutoff frequency		3057.7148437500005 Hz

Table 2.16: BPF0 values

Coefficient number	Continuous value	Fixed-point representation
0	-0.0009765625	0b1111111111100000
1	-0.006561279296875	0b1111111100101001
2	-0.00897216796875	0b1111111011011010
3	-0.009674072265625	0b1111111011000011
4	-0.053375244140625	0b1111100100101011
5	0.038787841796875	0b0000010011110111
6	0.2159423828125	0b1110010001011100
7	0.577667236328125	0b0100100111110001
8	0.577667236328125	0b0100100111110001
9	-0.2159423828125	0b1110010001011100
10	0.038787841796875	0b0000010011110111
11	-0.053375244140625	0b1111100100101011
12	0.009674072265625	0b1111111011000011
13	-0.00897216796875	0b1111111011011010
14	-0.006561279296875	0b1111111100101001
15	-0.0009765625	0b1111111111100000

Table 2.17: BPF1 values

Coefficient number	Continuous value	Fixed-point representation
0	-0.00018310546875	0b1111111111111010
1	0.0047607421875	0b0000000010011100
2	0.015625	0b0000001000000000
3	-0.013763427734375	0b1111111000111101
4	-0.117431640625	0b1111000011111000
5	-0.160491943359375	0b1110101101110101
6	0.020263671875	0b0000001010011000
7	0.275054931640625	0b0010001100110101
8	0.275054931640625	0b0010001100110101
9	0.020263671875	0b0000001010011000
10	-0.160491943359375	0b1110101101110101
11	-0.117431640625	0b1111000011111000
12	0.013763427734375	0b1111111000111101
13	0.015625	0b0000001000000000
14	0.0047607421875	0b0000000010011100
15	-0.00018310546875	0b1111111111111010

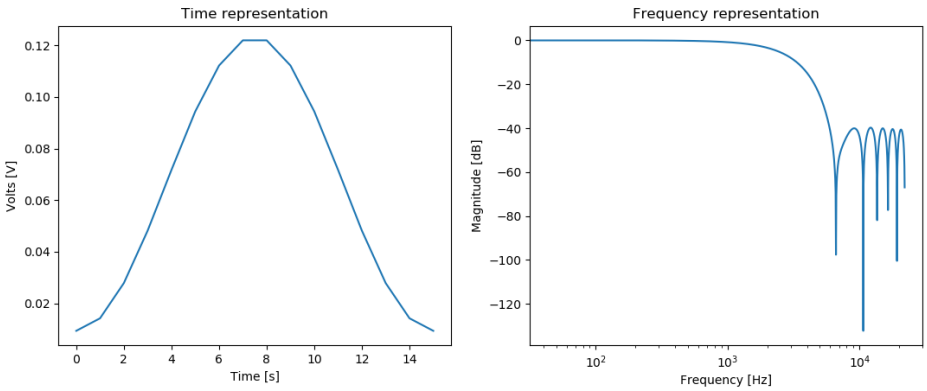

Figure 2.18: LPF0 (500 Hz) representation

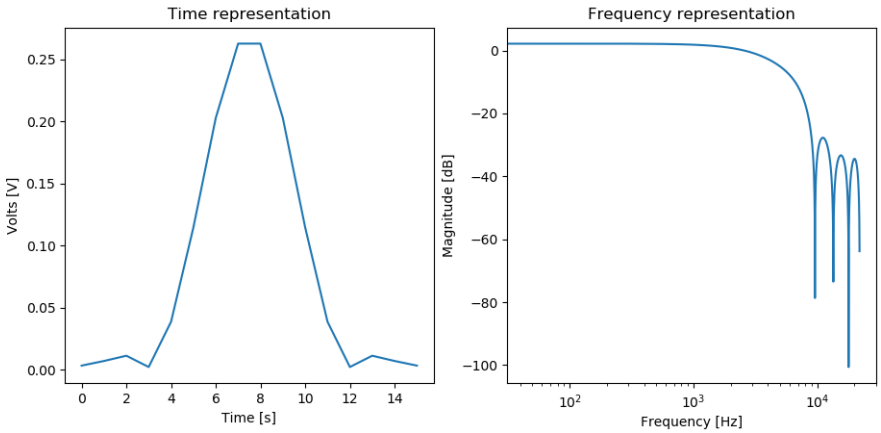

Figure 2.19: LPF1 (5 kHz) representation

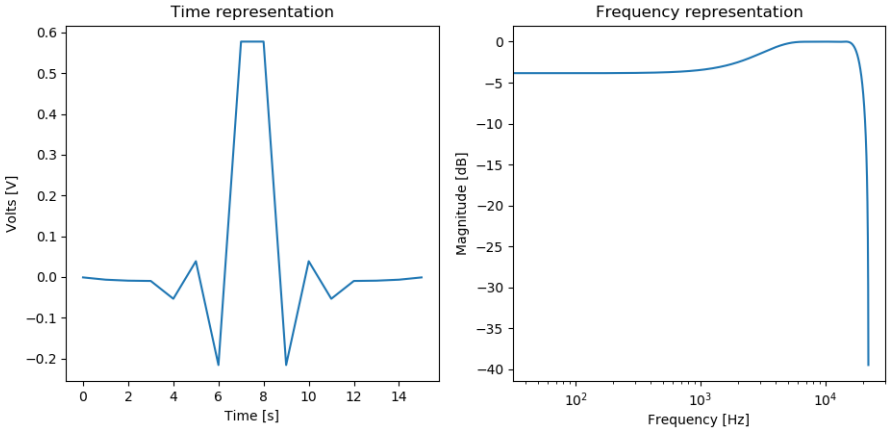

Figure 2.20: BPF0 (1 kHz - 20 kHz) representation

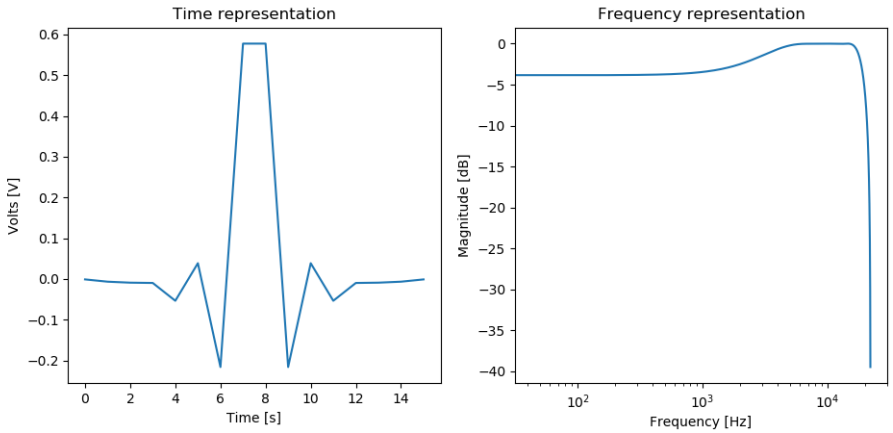

Figure 2.21: BPF0 (4 kHz - 10 kHz) representation

2.13 Basic modules

2.13.1 Register

Consists of a regular D-type flip, to be used as a temporal memory. The testbench of the module is shown in figure 2.22.
Inputs:

- clk: clock.
- data [WL : 0]: flip flop input.
- enable: flip flop write enable.
- rst: reset.
- sync_rst: resets the module in high state with a positive edge of clock.

Outputs:

- data_out [WL : 0]: flip flop output.

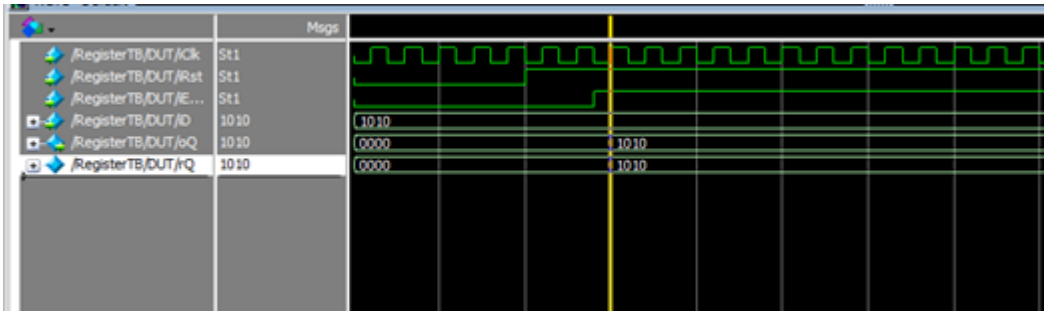

Figure 2.22: Register simulation

2.13.2 Register with initial conditions

Consists of a regular register but an initial condition is set with the reset signal is in low state. It will be used mainly for the register file for setting initial conditions in the filter flow. The testbench of the module is shown in figure 2.23.
Inputs:

- clk: clock.
- data [WL : 0]: flip flop input.
- enable: flip flop write enable.
- initial_condition [WL : 0]: default value when rst signal is low.
- rst: reset.
- sync_rst: resets the module in high state with a positive edge of clock.

Outputs:

- data_out [WL : 0]: flip flop output.

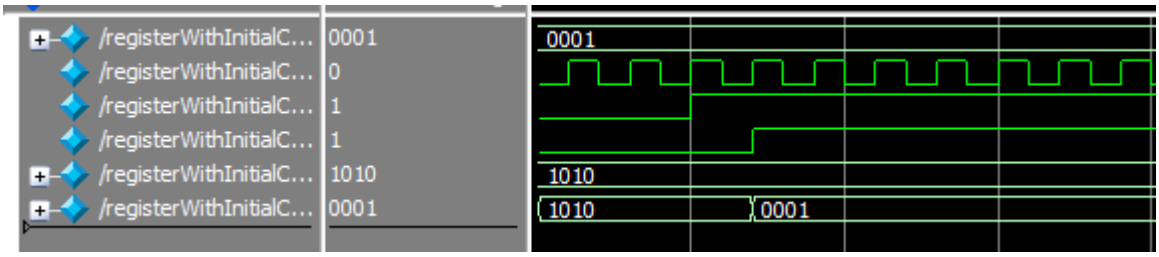

Figure 2.23: Register with initial condition simulation

2.13.3 Shift register

Consists of a chain of registers, whose outputs are the next register input, the output is the set of all register outputs, it has a dimension of $N^6 * WL^7$. It's used to save data over n iterations. The input is the data received from I2S bus and the output will be utilized in the fixed-point MAC. The testbench of the module is shown in figure 2.25 and the its respective block diagram in figure 2.24.
Inputs:

- clk: clock.
- data [WL : 0]: flip flop input.
- enable: flip flop write enable.
- rst: reset.

Outputs:

[6]Filter coefficients number
[7]Word Length

- data_out [WL : 0]: concatenated outputs of all registers.

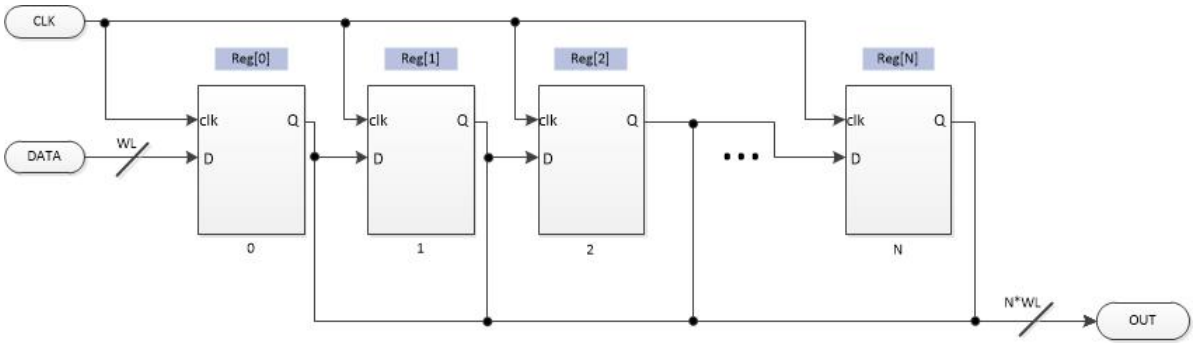

Figure 2.24: Register chain block diagram

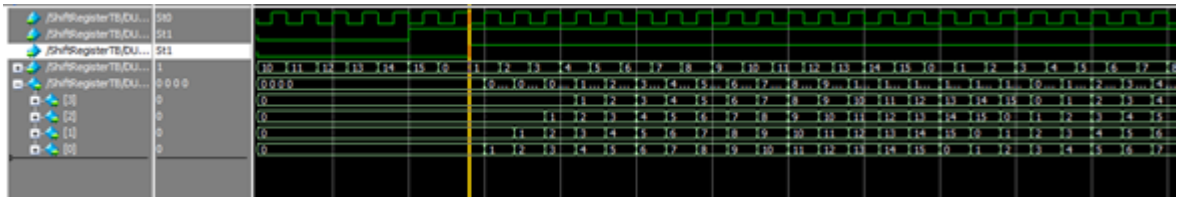

Figure 2.25: Register chain simulation

2.13.4 Serializer

Converts a data bus of "WL" dimension to subsequent pulses with I2S clock as reference. The output is to be connected to I2S_DAC. The testbench of the module is shown in figure 2.26.

Inputs:

- clk: clock.
- data_to_transmit [WL : 0]: data bus to be serialized.
- enable: saves the input data.
- rst: reset.
- shift: when there's a clock positive edge and this signal in high state a bit of the saved data is transmitted.

Outputs:

- Serial_Output.

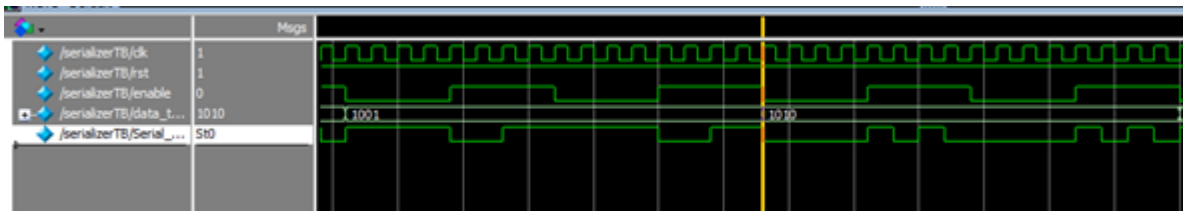

Figure 2.26: Serializer simulation

2.13.5 Left-Right decoder (LRDecoder)

Converts the serial data from I2S_ADC to a bus with WL dimension. It's basically a shift register with dimension 1 for each flip-flop. The testbench of the module is shown in figure 2.27.

Inputs:

- clk: clock.
- serialIn: serial data to be converted.
- rst: reset.
- sideSelector: enable to start converting the serial data.

Outputs:

- parallelOutput[WL : 0]: deserialized data.

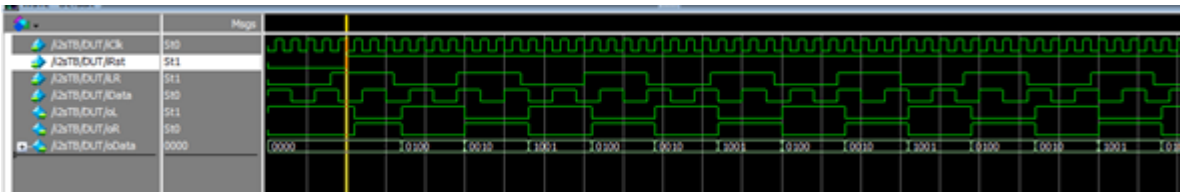

Figure 2.27: LRDecoder simulation

2.13.6 Multiplexers

Since a lot of multiplexers are used thorough all the project, specific size multiplexers are needed:

- 16 bit 16 to 1 multiplexer.
- 16 bit 8 to 1 multiplexer.
- 1 bit 3 to 1 multiplexer.
- 1 bit 2 to 1 multiplexer.

The testbench of the module is shown in figure 2.28.

Inputs:

- data [WL*N – 1 : 0]: concatenated input data.
- selector [log2(N) – 1 : 0]: serial data to be converted.

Outputs:

- out[WL : 0]: selected input.

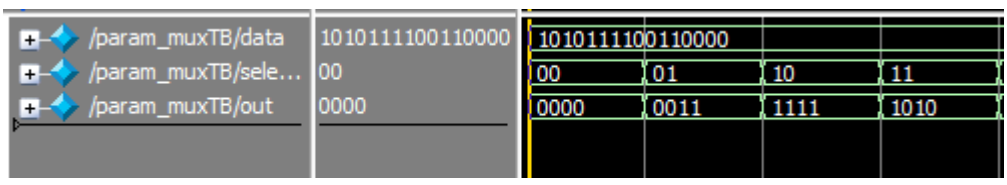

Figure 2.28: Multiplexer simulation

2.13.7 Single bit parametrized demultiplexer

Since is specified a single bit as output, it can be implemented as a shifter. It will be used for the register file module as an address-write enable converter. The testbench of the module is shown in figure 2.29.

Inputs:

- selector [log2(N) – 1 : 0]: selected address.

Outputs:

- dataOut[N - 1 : 0]: concatenated enables bus.

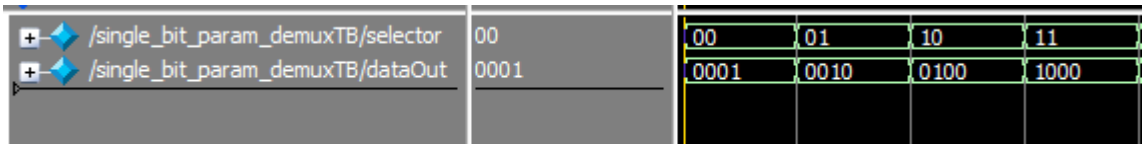

Figure 2.29: Single bit parametrized demultiplexer simulation

2.13.8 Counter

The system works as a serial filter, so almost all data will need to be stored and used periodically, this module will be used thorough a lot of modules in the project to synchronize the data flow. The testbench of the module is shown in figure 2.31 and the its respective block diagram in figure 2.30.

Inputs:

- clk: clock.
- count: add to current count with a positive edge of the clock.
- rst: reset.
- sync_rst: resets the module in high state with a positive edge of clock.

Outputs:

- countOut[log2(N) : 0]: current count.

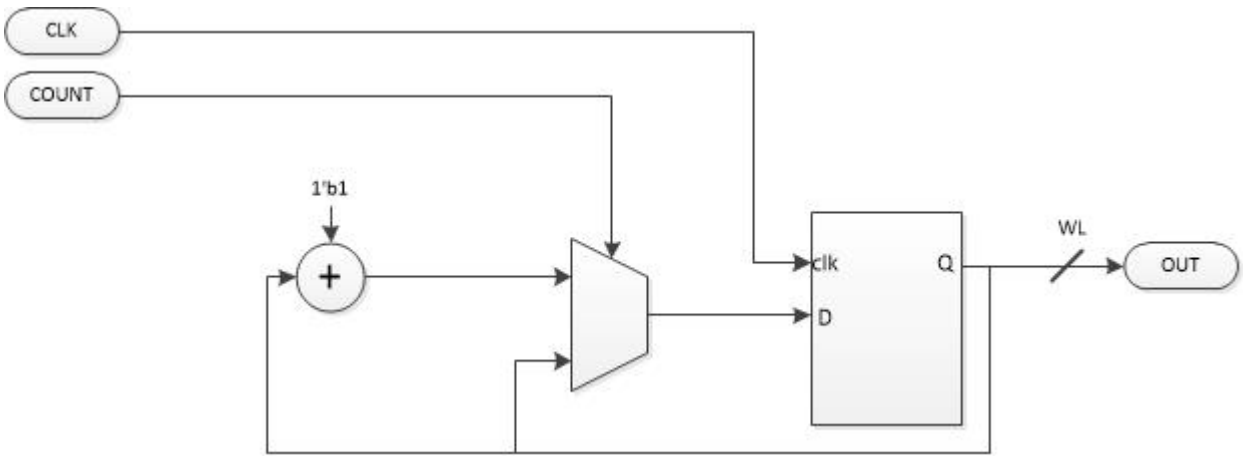

Figure 2.30: Counter block diagram

Figure 2.31: Counter simulation

2.13.9 Fixed-point MAC

Consists of a combinational signed multiplication module, is a part of the MAC sub-modules. The output is defined as equation 2.5.

$$D = A * B + C \tag{2.5}$$

The testbench of the module is shown in figure 2.32.
Inputs:

- A [WL – 1 : 0]: first data.
- B [WL – 1 : 0]: second data.
- C [WL – 1 : 0]: addition data.

Outputs:

- D [WL – 1 : 0] output data, defined as $A * B + C$.

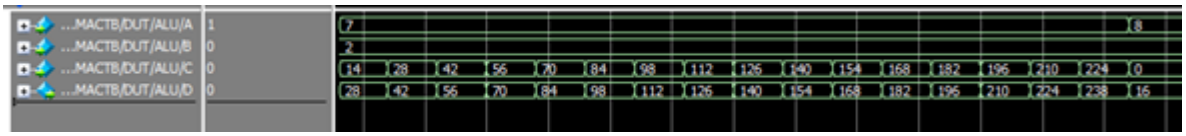

Figure 2.32: Fixed-point MAC simulation

2.13.10 Serial MAC

Is the serial multiplication module, performs a multiplication and an addition per clock cycle. This module is used to generate the convolution between the input data and the coefficients. It takes N cycles to get an output. The testbench of the module is shown in figure 2.34 and the its respective block diagram in figure 2.33.
Inputs:

- A [WL – 1 : 0]: first data.
- B [WL – 1 : 0]: second data.
- clk: clock.
- rst: reset.
- enable: performs a multiplication, addition and save when this signal is set to high and a positive edge on the clock occurs.

Outputs:

- result [WL – 1 : 0] output data, after WL clock pulses the data is ready.

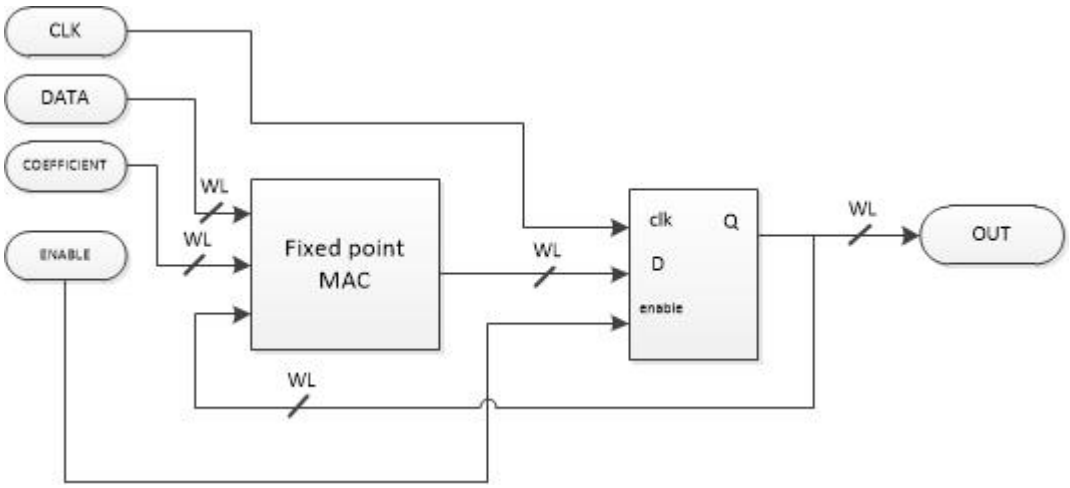

Figure 2.33: Serial MAC block diagram

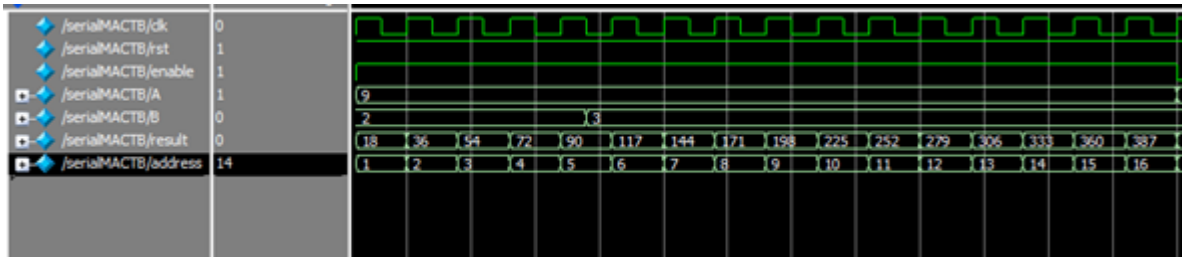

Figure 2.34: Serial MAC simulation

2.13.11 Division

A fixed-point division is needed to achieve the NLMS noise cancellation algorithm. The testbench of the module is shown in figure 2.35.

Inputs:

- A [WL – 1 : 0]: first data.
- B [WL – 1 : 0]: second data.

Outputs:

- D [WL – 1 : 0] output data, defined as $\frac{A}{B}$

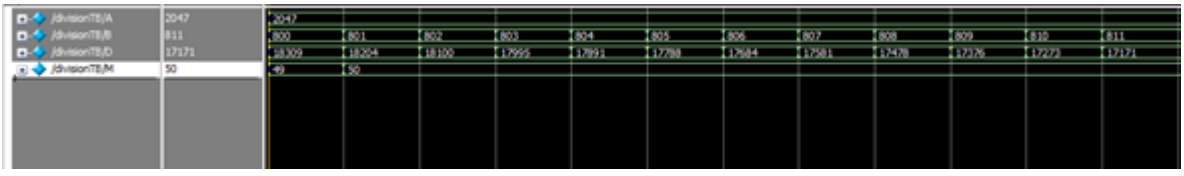

Figure 2.35: Division simulation

2.13.12 Serial delay

When using in regular FIR filter mode, each channel is filtered. When the adaptive filter mode is used the left channel has only the noise to cancel; based on both channels the new coefficients are calculated. The serial delay module function as a repeater to mirror the output to the left channel and generate a stereo output. This module is essentially a register chain with N registers.
Inputs:

- clk: clock.
- rst: reset.
- signal: serial data to be repeated after WL clock pulses.

Outputs:

- delayed_signal.

2.13.13 One shot

Converts the left/right signal from I2S bus to a single pulse. This module is used to trigger the register chain module to push once every N clocks the new data into the chain; it's only needed to register the left/right signal.
Inputs:

- iClk: clock.
- iRst: reset.
- iSignal: signal to convert into a single pulse.

Outputs:

- oOneShot: single pulse output.

2.13.14 New coefficient

For noise cancellation purposes new filter coefficients need to be calculated for both LMS or NLMS adaptive filtering. This module generates the $w(n) + \frac{\mu x(n)e(n)}{||x(n)||^2}$ or $w(n) + \mu x(n)e(n)$ operations, depending on which noise cancellation method is used (NLMS or LMS respectively). The testbench of the module is shown in figure 2.36.
Inputs:

- error [WL – 1 : 0]: difference between input and desired signal.
- AFM: decides from either LMS or NLMS algorithm.
- x_in [WL – 1 : 0]: data from the input buffer.
- w [WL – 1 : 0]: previous coefficient.
- SNC [WL – 1 : 0]: numerator sensitivity constant.

- SDC [WL – 1 : 0]: denominator sensitivity constant.
- x_power [WL – 1 : 0]: energy of the input signal.

Outputs:

- result [WL – 1 : 0]: new coefficient to be written.

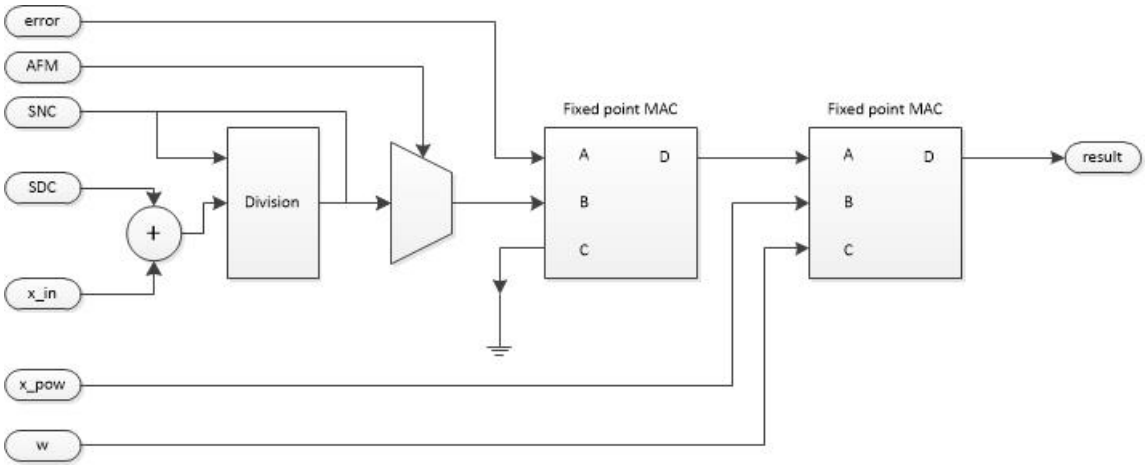

Figure 2.36: New coefficient simulation

2.13.15 Serial ROM

The whole project was developed without the use of ROM or RAM memories, instead register/multiplexer realizations were developed. This module consists of a $WL * N$ width data with all the coefficients, this bus is connected into a parametrized multiplexer. The testbench of the module is shown in figure 2.37.
Inputs:

- addr [log2(WL) – 1 : 0]: memory address to read.

Outputs:

- q[WL - 1 : 0]: data in specified address.

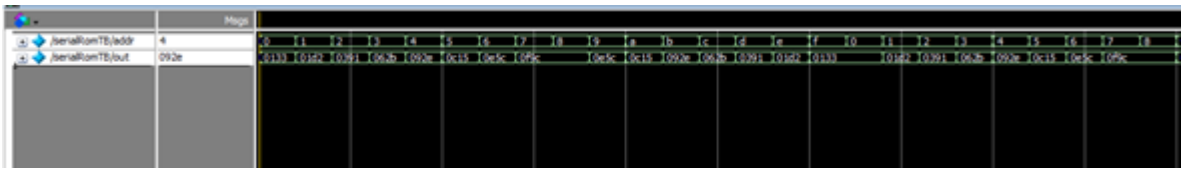

Figure 2.37: Serial ROM simulation

2.13.16 Simple dual port RAM single clock

The whole project was developed without the use of ROM or RAM memories, instead register/multiplexer realizations were developed. This module consists of a parametrized demultiplexer to drive the write enable to the selected register by the

write address. In the output there's a parametrized multiplexer after the registers, it chooses the output by the read address value. The testbench of the module is shown in figure 2.39 and the its respective block diagram in figure 2.38.

Inputs:

- clk: clock.
- rst: reset.
- we: write enable.
- data [WL : 0]: flip flop input.
- read_addr [N : 0]: data output selector.
- write_addr [N : 0]: data write address.

Outputs:

- q [WL : 0]: combinational read data.

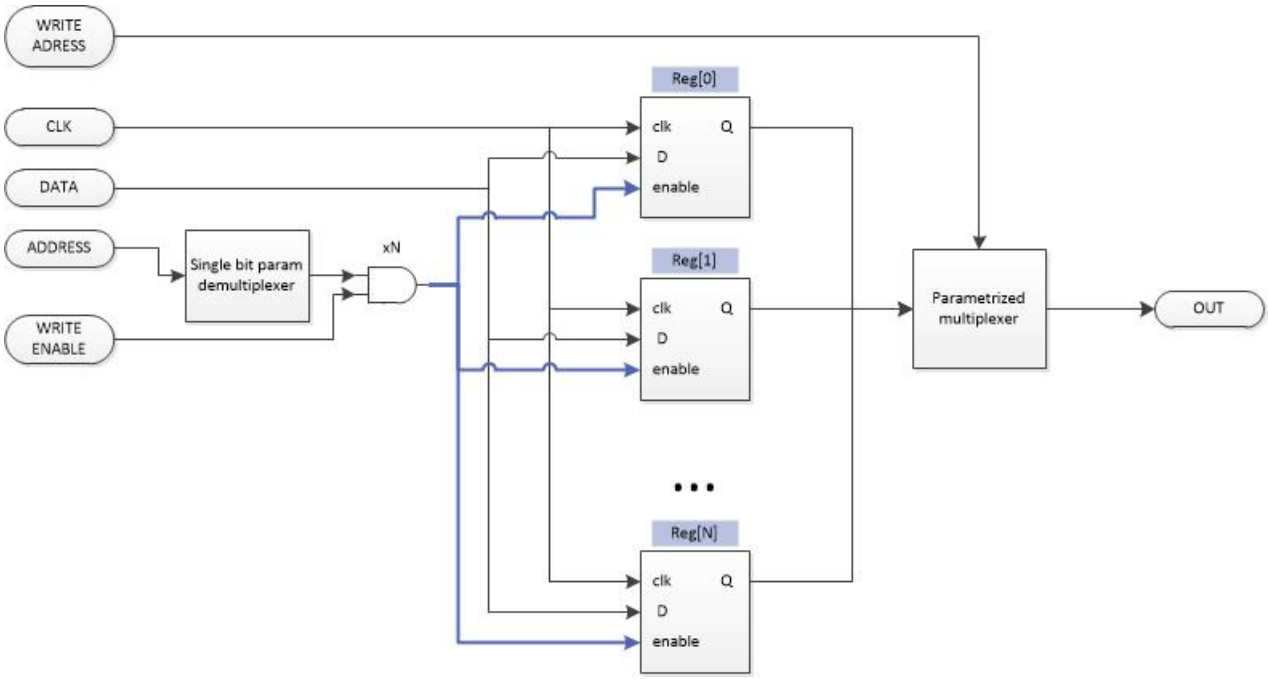

Figure 2.38: Serial RAM block diagram

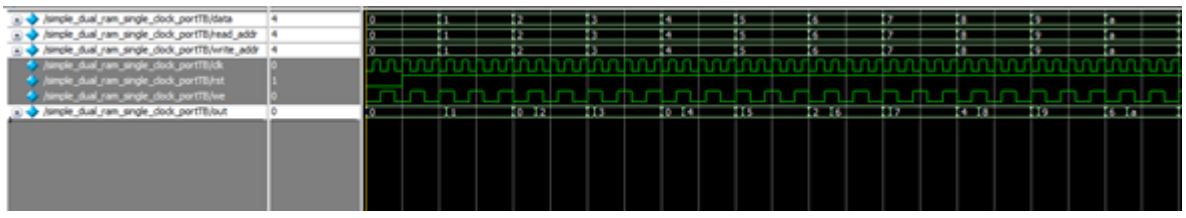

Figure 2.39: Serial RAM simulation

2.13.17 I2C

This module consists of 3 internal modules: a register file, an I2C receiver, and a i2c state machine. The block diagram is shown in figure 2.40.
Inputs:

- clk: clock.
- rst: reset.
- SCL: I2C bus serial clock.

Inouts:

- SDA: I2C bus serial data.

Outputs:

- UDF0 [WL*N – 1 : 0]: user defined filter 0 concatenated data.
- UDF1 [WL*N – 1 : 0]: user defined filter 1 concatenated data.
- AFM: adaptive filter mode selector.
- FOPT [2 : 0]: filter type selector.
- OUTC [1 : 0]: output configuration.
- SNC [WL – 1 : 0]: numerator sensitivity constant.
- SDC [WL – 1 : 0]: denominator sensitivity constant.
- CS: left and right channel selector/swapper.

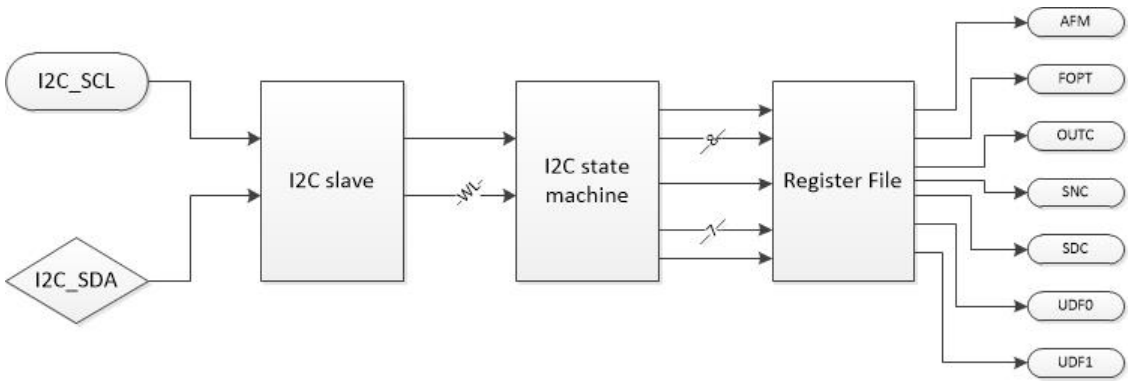

Figure 2.40: I2C block diagram

I2C slave [8]

The only objective of this module is to concatenate the serial in data and output a pulse when the data is ready if the slave address matched, sending also an ACK bit. The data flow will be managed in the state machine.

[8]This module couldn't be simulated because of the inout port, it will be verified in the FPGA implementation.

I2C state machine

This module gets as inputs the outputs of the I2C slave module, it controls the flow to write into the register file. The testbench of the module is shown in figure 2.42 and the its respective block diagram representation in figure 2.41.

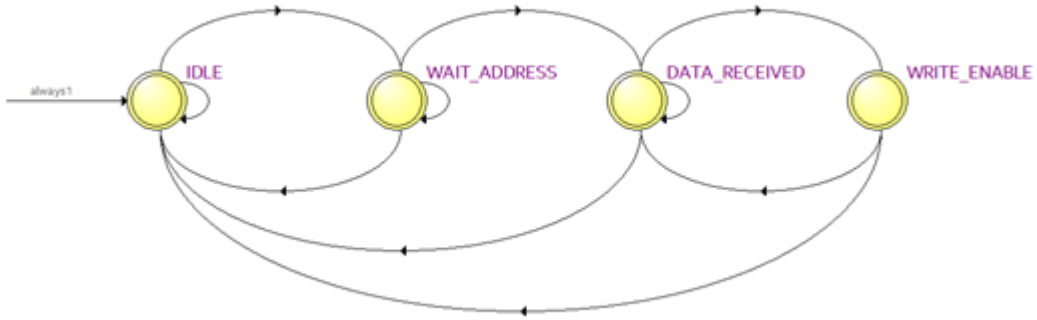

Figure 2.41: I2C state machine block diagram

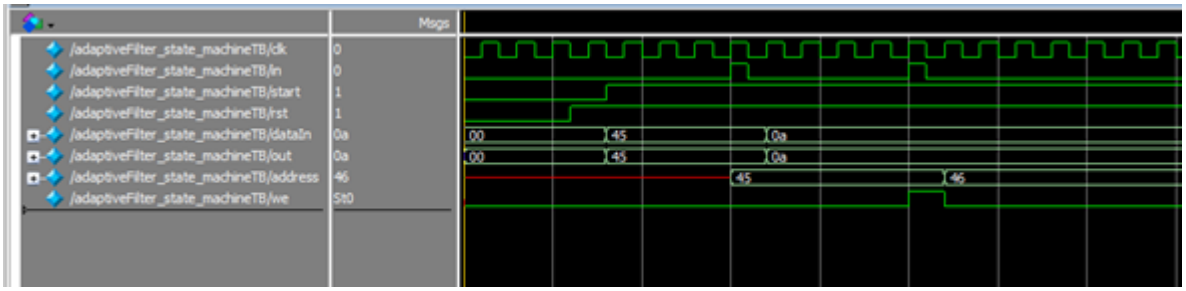

Figure 2.42: I2C state machine simulation

Register file

This module has the same structure as the Simple dual port RAM single clock module, the only change is the registers used, this one is built with the Register with initial conditions instead, to set the control variables to a fixed value once the reset is deserted. The outputs are concatenated and initialized according to the Memory map section.

FPGA implementation

3.14 Modules verification

For the following modules, a RTL representation and the module verification made in signal tap can be found. Some modules won't have RTL representation since they are too big and its elements can't be recognized.

3.14.1 Register

Figure 3.43 shows the synthesized RTL, conformed by a flip flop and a mux that either writes the data to the output or a zero if activated by the sync_rst signal. The functionality is demonstrated in figure 3.44, where the input goes to the output with the positive edge of clk. This module is used as a general D-flip flop thorough all the project.

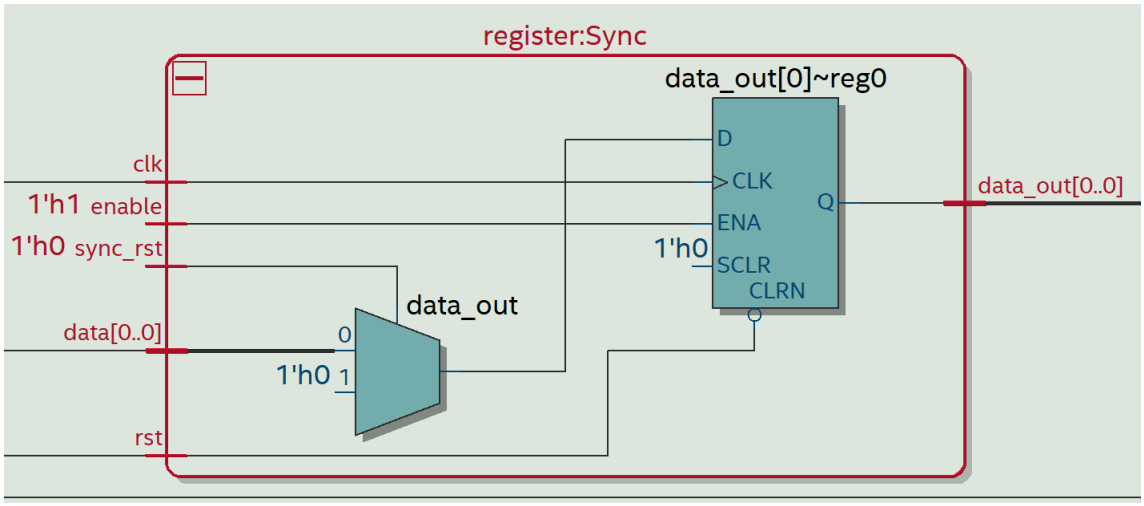

Figure 3.43: Register RTL

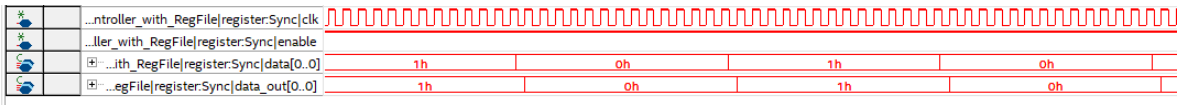

Figure 3.44: Register signal tap verification

3.14.2 Register with initial conditions

Figure 3.45 shows the synthesized RTL, conformed by a flip flop and a mux that either writes the data to the output or a zero if activated by the sync_rst signal.

Additionally the intial conditions is written to the D-flip flop as the default value, when rst signal is low. The functionality is demonstrated in figure 3.46, where the input goes to the initial condition when rst is not asserted and the output with the positive edge of clk. This module is used in the register file to set a default mode in the system.

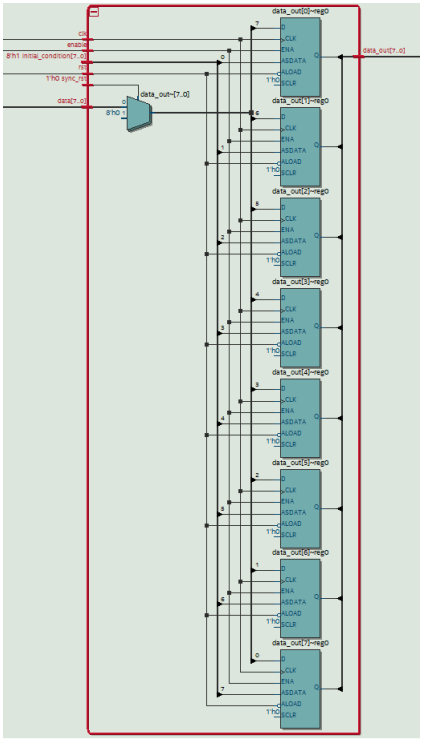

Figure 3.45: Register with initial condition RTL

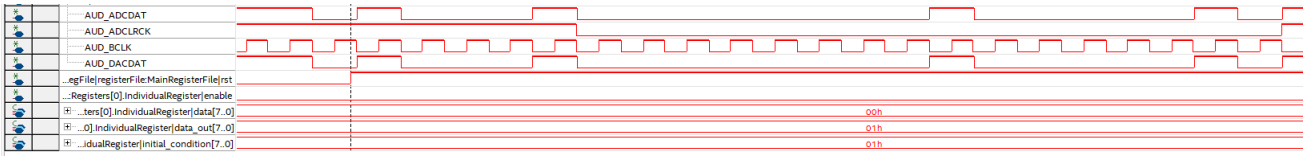

Figure 3.46: Register with initial condition signal tap verification

3.14.3 Shift register

The functionality is demonstrated in figure 3.47, where the parallel input gets pushed through the data array with each LR pulse.

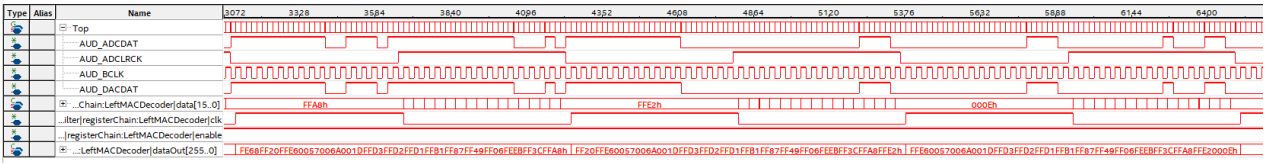

Figure 3.47: Shift register signal tap verification

3.14.4 Serializer

Figure 3.48 shows the synthesized RTL, conformed by shift register of 1 bit that shifts the data bits per clk edge, serializing the data. The functionality is demonstrated in figure 3.49, where the input data gets serialized based on the clk.

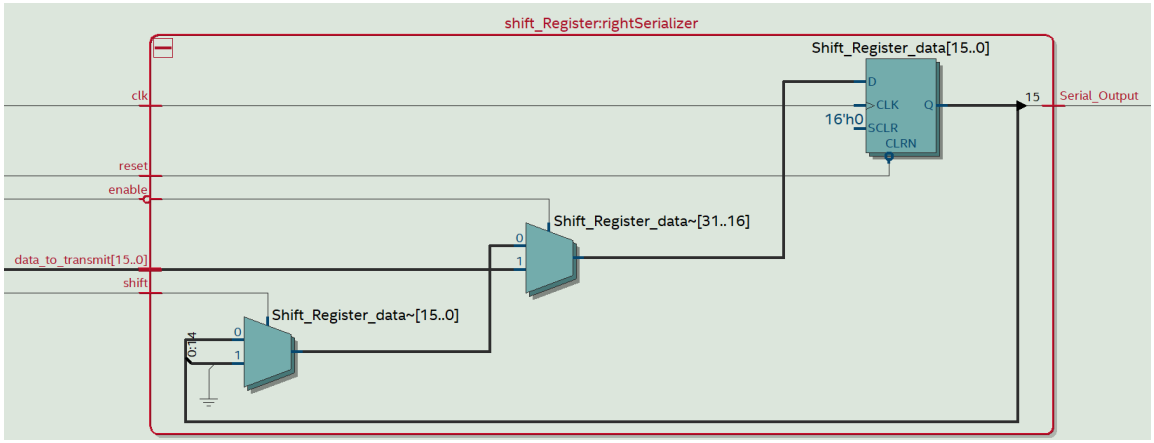

Figure 3.48: Serializer RTL

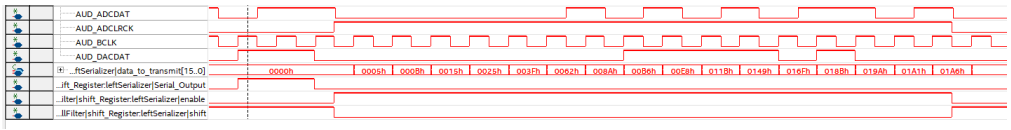

Figure 3.49: Serializer signal tap verification

3.14.5 Left-Right decoder

Figure 3.50 shows the synthesized RTL, conformed by a shift register of 1 bit that concatenates the input serial bits. The functionality is demonstrated in figure 3.51, where the input serial data outputs as its parallel bus representation.

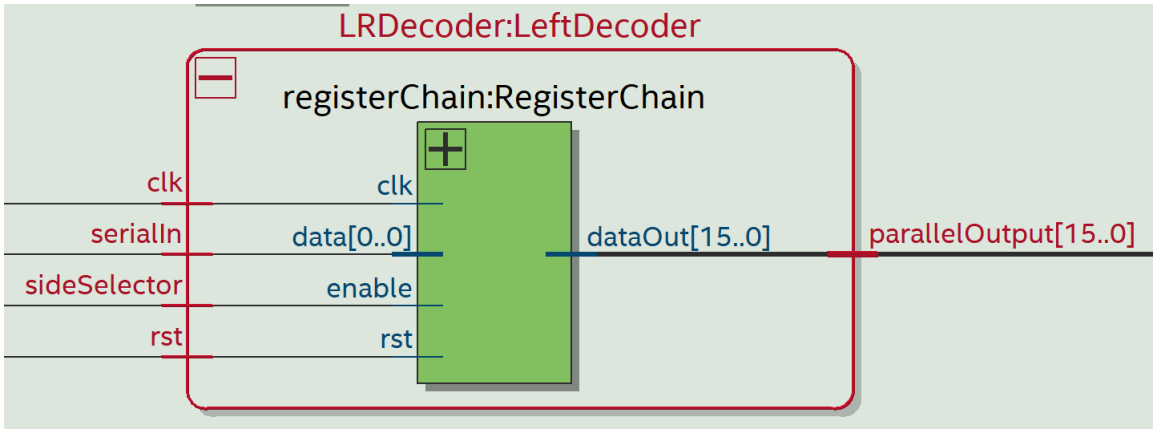

Figure 3.50: Left-Right decoder RTL

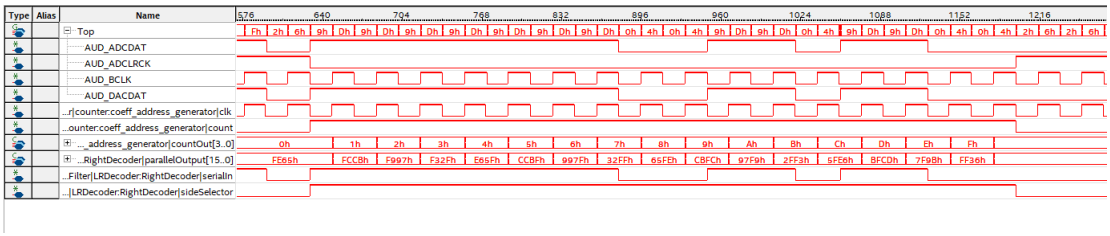

Figure 3.51: Left-Right decoder signal tap verification

3.14.6 Multiplexers

Figure 3.52 shows the synthesized RTL, conformed by general multiplexer controlled by a decoder as the mux selector, the input data is parallel and gets separated evenly into the multiplexer inputs.

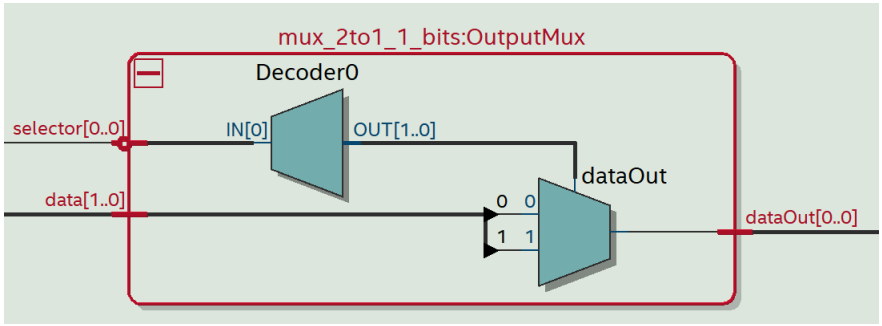

Figure 3.52: Multiplexer RTL

3.14.7 Single bit parametrized multiplexer

The functionality is demonstrated in figure 3.53, the signal tap diagram shows a 4 bit multiplexer, the selector is incremented per clk pulse and the output gets shifted.

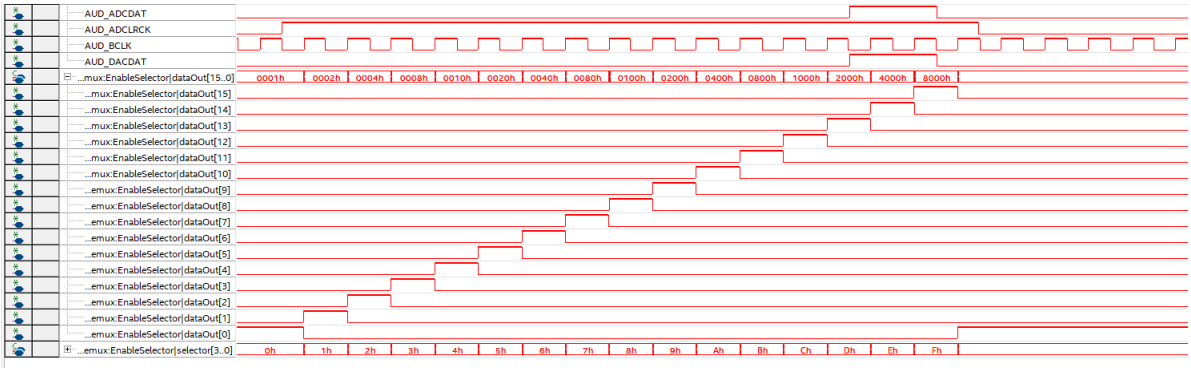

Figure 3.53: Single bit multiplexer signal tap verification

3.14.8 Counter

Figure 3.54 shows the synthesized RTL, conformed by a flip flop with feedback with the output + 1, which increments by each positive edge of the clock; the second multiplexer corresponds to the sync_rst function. The functionality is demonstrated

in figure 3.55, the output gets incremented with the clk and the count enable signal, it's cleared after through the sync_rst signal.

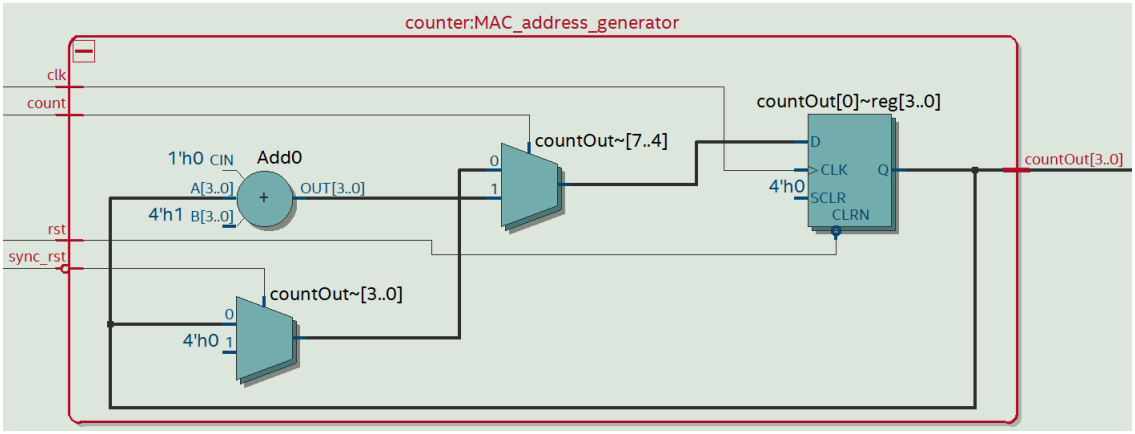

Figure 3.54: Counter RTL

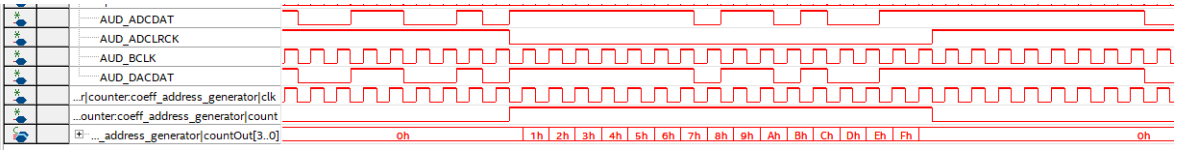

Figure 3.55: Counter signal tap verification

3.14.9 Fixed-point MAC

Figure 3.56 shows the synthesized RTL, conformed by a signed multiplier connected to an adder. The functionality is demonstrated in figure 3.57 (note that the input and output data are represented in fixed point format; Q1.15).

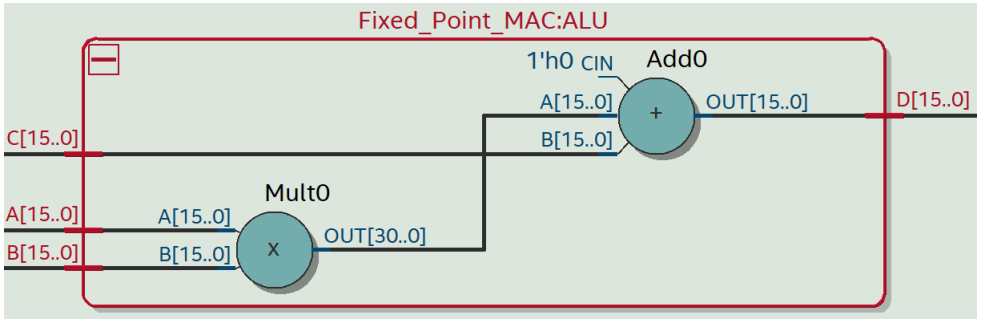

Figure 3.56: Fixed-point MAC RTL

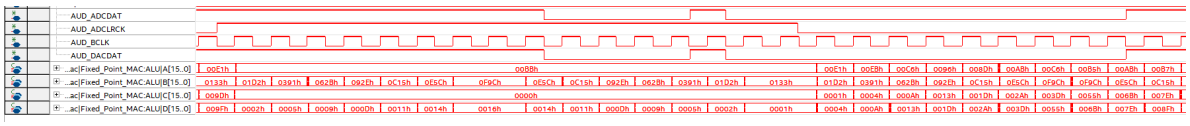

Figure 3.57: Fixed-point MAC signal tap verification

3.14.10 Serial MAC

Figure 3.58 shows the synthesized RTL, conformed by the fixed-point MAC module with feedback from the output to the addition input port. The functionality is demonstrated in figure 3.59, per each positive edge of the clock the data gets multiplied and added to the output, resulting in $y(x) = \sum_{n=0}^{15} w(n)x(n)$.

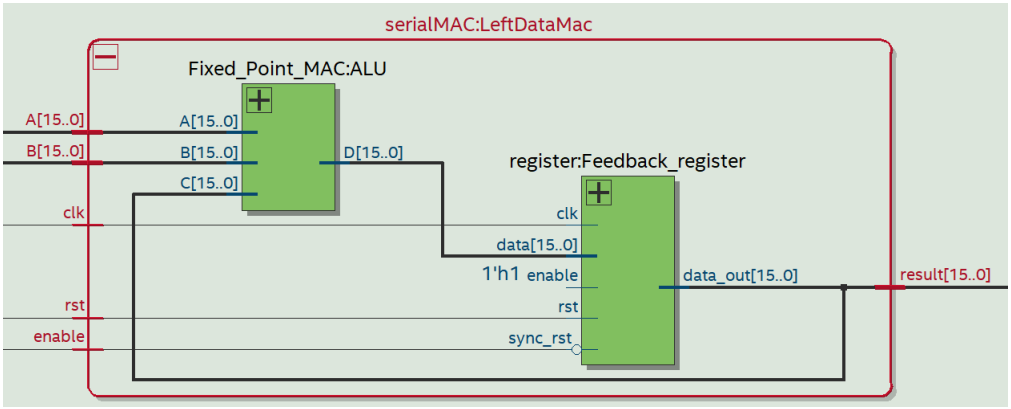

Figure 3.58: Serial MAC RTL

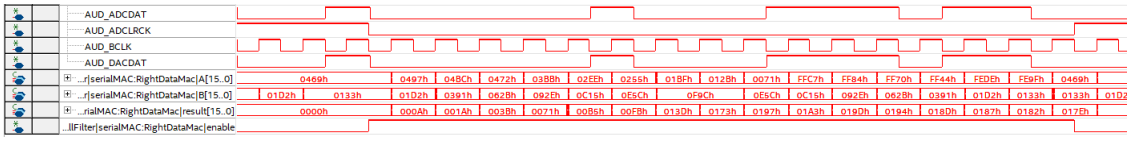

Figure 3.59: Serial MAC signal tap verification

3.14.11 Division

The functionality is demonstrated in figure 3.60, the data gets divided in fixed-point format (Q1.15).

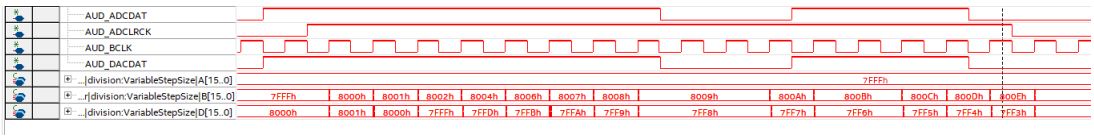

Figure 3.60: Division signal tap verification

3.14.12 Serial delay

The functionality is demonstrated in figure 3.61, the output is delayed 16 positive edges of the clk signal from the input.

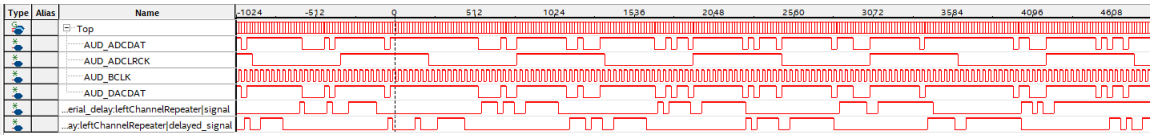

Figure 3.61: Serial delay signal tap verification

3.14.13 One shot

Figure 3.62 shows the synthesized RTL, interpreted as a simple flip flop. The functionality is demonstrated in figure 3.63, the input signal gets processed as a single pulse based on the channel selector signal.

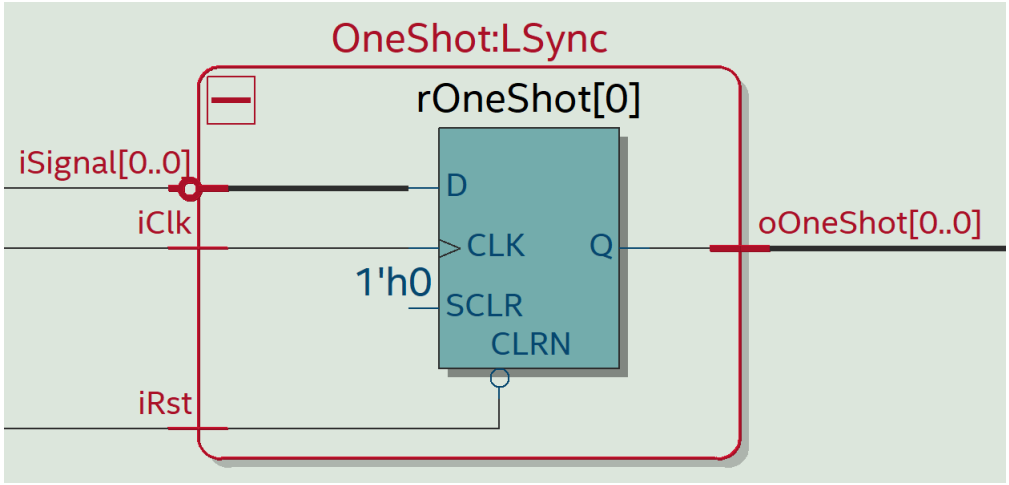

Figure 3.62: One shot RTL

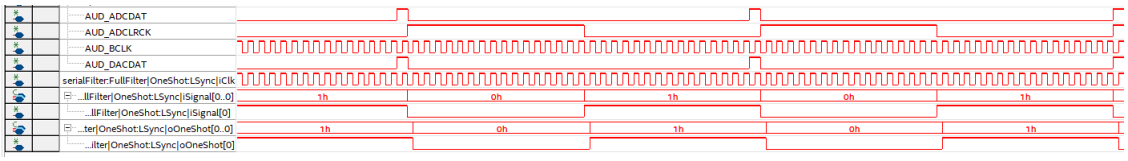

Figure 3.63: One shot signal tap verification

3.14.14 New coefficient

Figure 3.64 shows the synthesized RTL, conformed by two multipliers and a division module. The functionality is demonstrated in figure 3.65, the input signal gets processed through the multipliers and division modules resulting in the operation: $w(n+1) = w(n) + \frac{\mu x(n) e(n)}{||x(n)||^2}$ in fixed-point format, Q1.15.

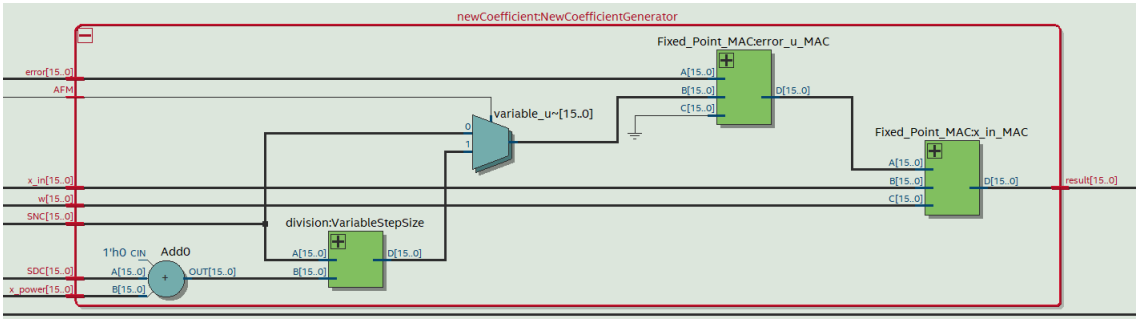

Figure 3.64: New coefficient RTL

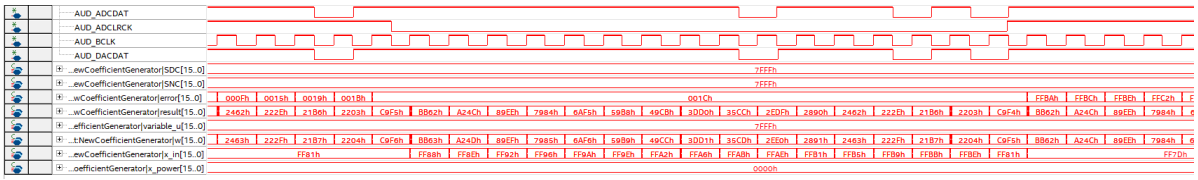

Figure 3.65: New coefficient signal tap verification

3.14.15 Serial ROM

Figure 3.66 shows the synthesized RTL, conformed by a simple multiplexer controlled by the address. The functionality is demonstrated in figure 3.67, going through all the input data increasing the address step by step.

Figure 3.66: Serial ROM RTL

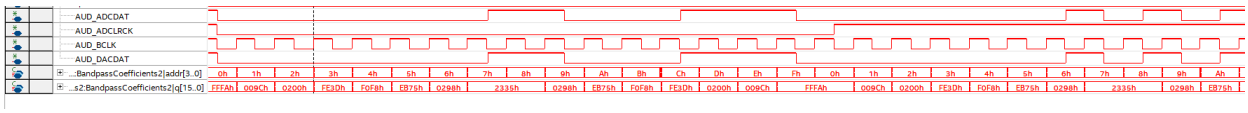

Figure 3.67: Serial ROM signal tap verification

3.14.16 Simple dual port RAM single clock

The functionality is demonstrated in figure 3.68, going through all the input data increasing the address step by step and also being written depending on the we (write enable) signal.

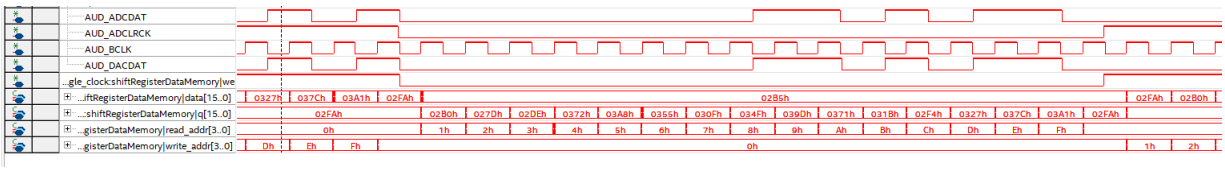

Figure 3.68: Simple dual port RAM single clock signal tap verification

3.14.17 I2C

Figure 3.69 shows the synthesized RTL with the i2c_slave, adaptiveFilter_state_machine and registerFile interconnections. An additional module register is added to sync the two I2S_CLK and I2C_SCL domain clocks.

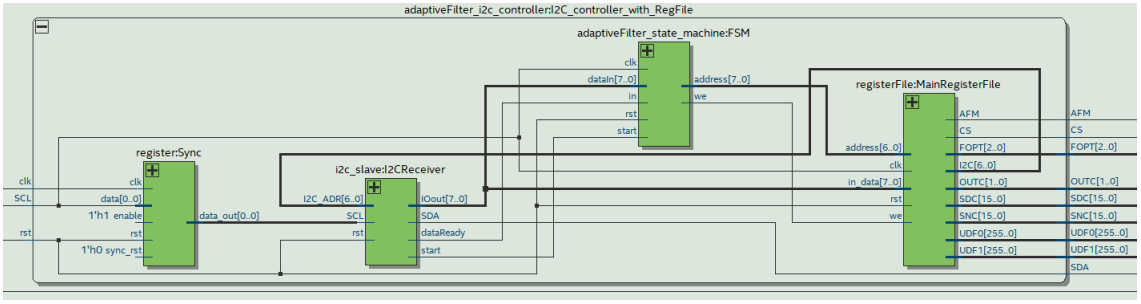

Figure 3.69: I2C RTL

I2C slave

The functionality is demonstrated in figure 3.70, it shows a simple stream of I2C ending it with the ACK bit sent by the module.

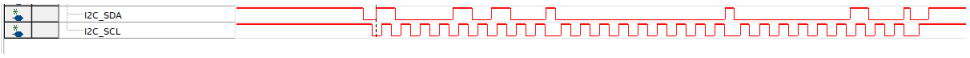

Figure 3.70: I2C signal tap verification

I2C state machine

Figure 3.71 shows the synthesized RTL. The functionality is demonstrated in figure 3.72, going through the I2C stream and showing the states of the state machine over the time and ending it with the ACK bit.

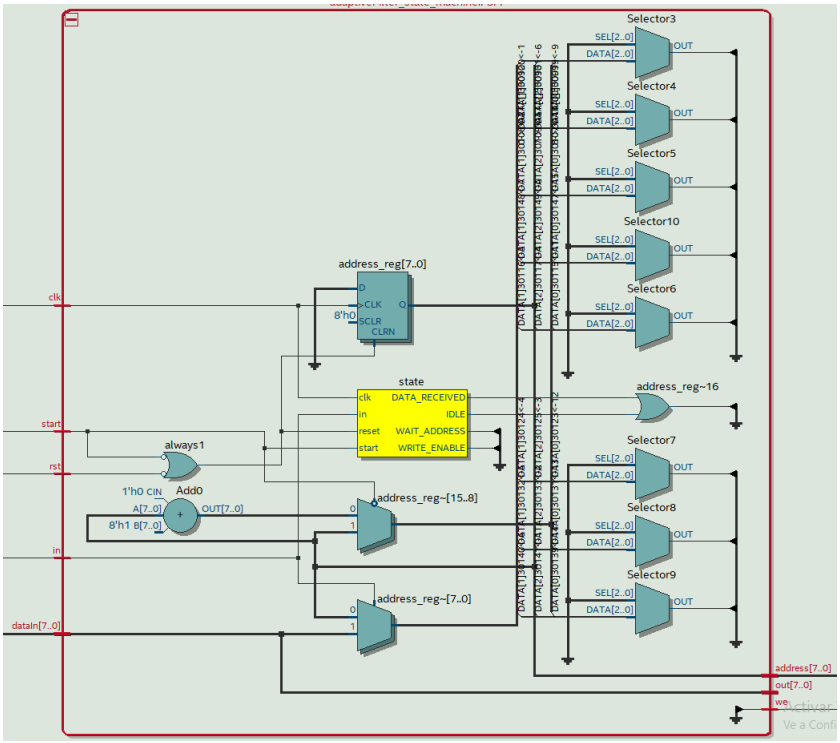

Figure 3.71: I2C state machine RTL

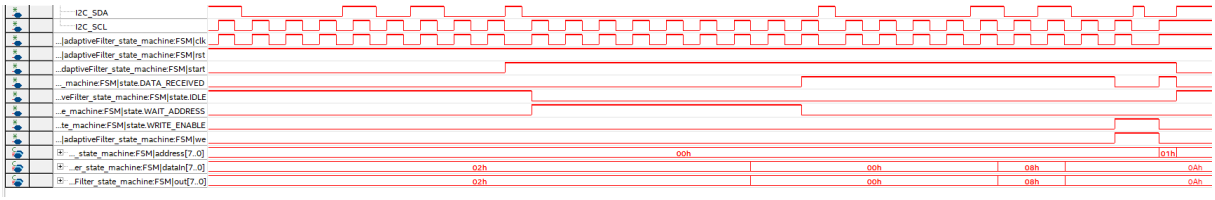

Figure 3.72: I2C state machine signal tap verification

Register file

The functionality is demonstrated in figure 3.73 over the I2C stream; at every time the register file is being read controlling the system flow, at the end a single data is being written into a specific register.

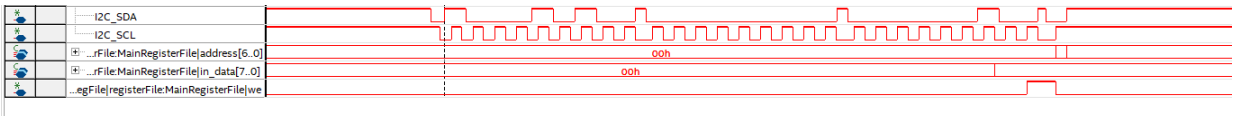

Figure 3.73: Register file signal tap verification

3.15 Synthesis summary

Figure 3.74 shows the required logic elements, registers, multipliers and pins that form the project.

Flow Status	Successful - Mon Aug 19 00:44:02 2019
Quartus Prime Version	18.1.0 Build 625 09/12/2018 SJ Lite Edition
Revision Name	Filtering
Top-level Entity Name	Filtering
Family	Cyclone IV E
Device	EP4CE115F29C7
Timing Models	Final
Total logic elements	3,518 / 114,480 (3 %)
Total registers	1884
Total pins	12 / 529 (2 %)
Total virtual pins	0
Total memory bits	0 / 3,981,312 (0 %)
Embedded Multiplier 9-bit elements	10 / 532 (2 %)

Figure 3.74: Quartus synthesis summary

3.16 Synthesis timing analysis

The information shown in figure 3.75 were used in the VLSI implementation as timing requirements.

	Fmax	Restricted Fmax	Clock Name
1	8.33 MHz	8.33 MHz	AUDIO_DAC:u4|oAUD_BCK
2	104.43 MHz	104.43 MHz	serialFilter:FullFilter|adaptiveFilter_i2c_contr...ontroller_with_RegFile|register:Sync|data_out[0]
3	155.64 MHz	155.64 MHz	CLOCK_50
4	194.33 MHz	194.33 MHz	PLL|altpll_component|pll|clk[0]
5	202.88 MHz	202.88 MHz	I2C_SCL
6	291.8 MHz	291.8 MHz	I2C_Module:I2C|mI2C_CTRL_CLK
7	599.88 MHz	437.64 MHz	serialFilter:FullFilter|OneShot:LSync|rOneShot[0]
8	628.14 MHz	437.64 MHz	serialFilter:FullFilter|OneShot:RSync|rOneShot[0]

Figure 3.75: Quartus timing analysis

3.17 Functional verification

Each one of the filter output modes were tested (shown in table 4.19) connecting a development board as I2C master, further information in *Development board software for verification* chapter. The tests are described down below.

- FIR filters: write MCR to use LPF0, LPF1, BPF0, BPF1, bypass and turn off output. Ensure the auditive output goes according the selected filter. Result: **OK**.
- Writable FIR filters: write both writable filters by I2C and select them as outputs. Ensure the auditive output goes according the selected filter. Result: **OK**.
- Adaptive filters: select both LMS and NLMS adaptive filter modes and filter the signal according to figures 2.16 and 2.17. Result: **Sine wave filtering: OK, Song noise filtering: Not OK**[9].

[9]Since the word length for the project is fixed to 16 bits, the new coefficients of the filter can't get the needed resolution to maintain the stability. At certain times the algorithm cancellates the noise in the song and suddenly it becomes unstable when the power of the input signal changes suddely; this behavior was confirmed by limiting the new coefficients to 16 bits in the reference model.

Development board software for verification

Since the system is designed to have as user input only the reset signal and the I2C lines, the NXP tool *Processor Expert* to generate a I2C master. The development board used is a Freescale FRDM-KL25Z. The main code can be found in 8.26.

4.18 User interface configuration

The I2C message construction is shown in 4.18.

- Start bit: transition from high to low level.
- Slave address: consists of a 7-bit word by default 0x45, can be rewritten but as the reset signal asserts a low level it returns to the default value.
- Read/Write: low level if read operation is selected, high level if write operation is selected. This module is a write only module, if read operation is selected it will be ignored.
- Data: after the slave address N data can be written in chain.

Table 4.18: I2C message constuction

Start bit	Slave address	Read/Write	Data 0	...	Data N	Stop bit

4.18.1 Main control register configuration

Table 4.19 shows the fixed values to be written in 0x0 address to configure the filter output. These values are used on the verification software.

Table 4.19: Main control register user configuration

Configuration	**Binary value**	**Hexadecimal value**
Low-pass filter 1	xxx00010	0x2
Low-pass filter 2	xxx00110	0x6
Band-pass filter 1	xxx01010	0xA
Band-pass filter 2	xxx01110	0xE
User-defined filter 1	xxx10010	0x12
Least mean square	xx011010	0x1A
Normalized least mean square	xx111010	0x3A
Bypass	xxxxxx01	0x1
Off	xxxxxxx0	0x0

Silicon implementation

5.19 Pad assignation

Due to the size of the project 48 pads were needed to fit the whole project. Figure 5.76 shows the pads location; the types of pads are separated by colors. Each type of pad are described below.

- vdd$n$: supply voltage for the core logic.
- vss$n$: supply ground for the core logic.
- vddpst$n$: supply voltage for IO drivers.
- vsspst$n$: supply ground for IO drivers.
- GndCon: specific pad for vsspst$n$ and vss$n$ connection.
- vddpstpoc: generates a delay for the supply voltage core to compensate the stability time for the IO drivers.
- signals: colored as green, correspond to the inputs, inouts and output described in section 2.8.
- corners: colored as black, are the edges of the pads noted with two letters refering to: north-east, north-west, south-west and south-east.

Figure 5.76: Pad assignation

The signal pads were located intendedly in the middle of each side, because the distance from the pads to the physical pins of the chip is minimum, securing that there will be no problems on the most critical signals of the system.

5.20 VLSI phases

Below each phase of the VLSI phase are reported. Further information on scripts used can be found in Annexes (Section 8.28.8).

5.20.1 Pad generation

The figure 5.77 shows the result of the mapping phase.

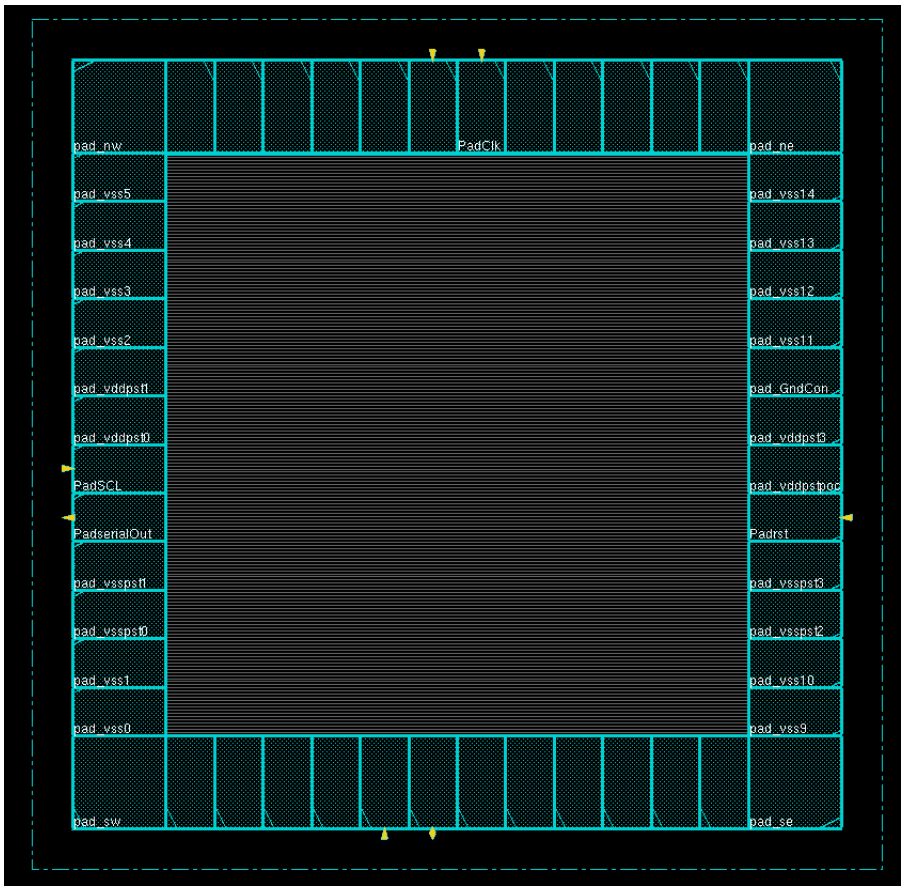

Figure 5.77: Pads

5.20.2 Powering and grid

The figure 5.78 shows the result of the powering and grid phase.

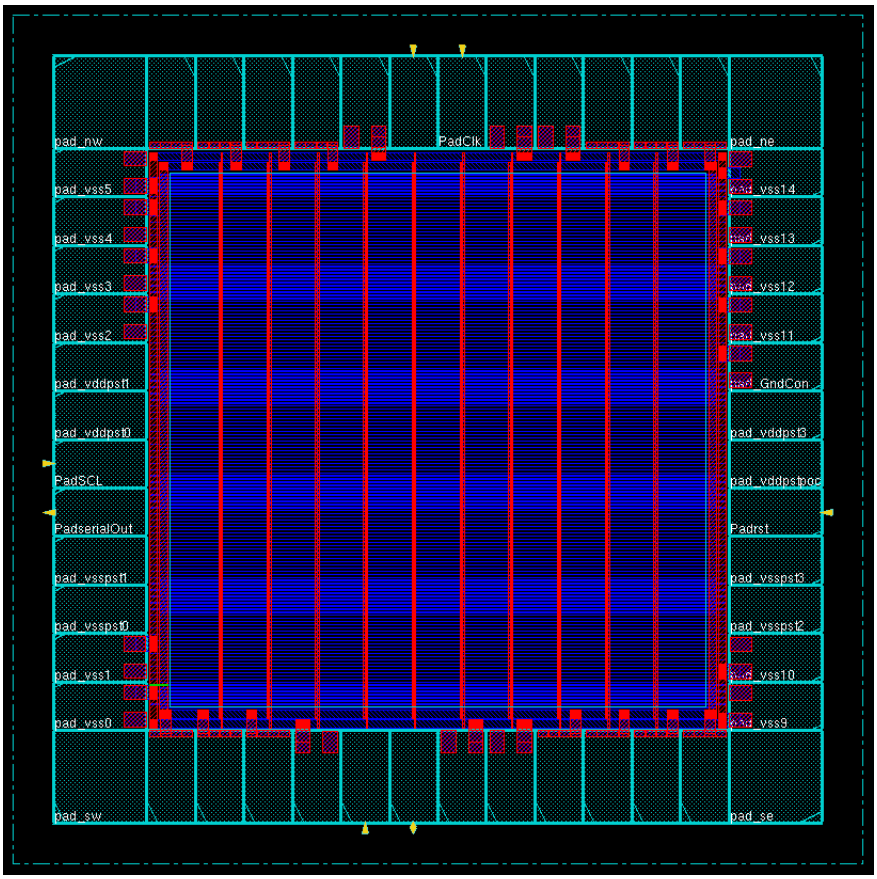

Figure 5.78: Powering and grid

5.20.3 Placement

The figure 5.79 shows the result of the placement phase.

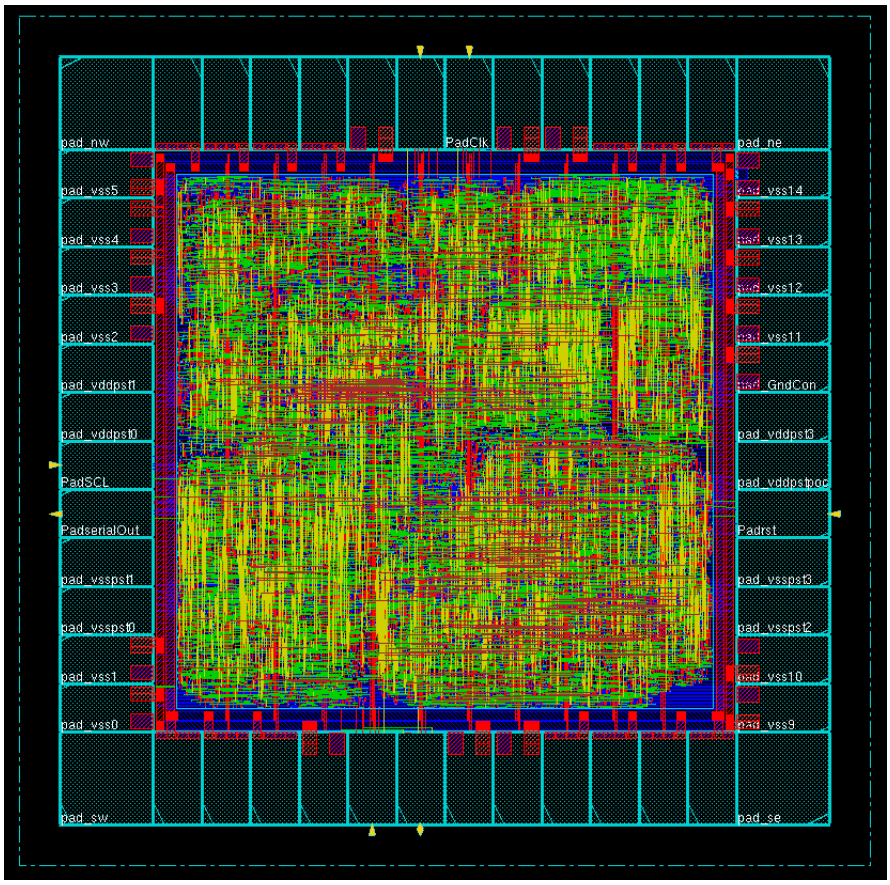

Figure 5.79: Placement

5.20.4 Placement with clock tree

The figure 5.80 shows the result of the placement with clock tree phase. The clock signal is represented in white color.

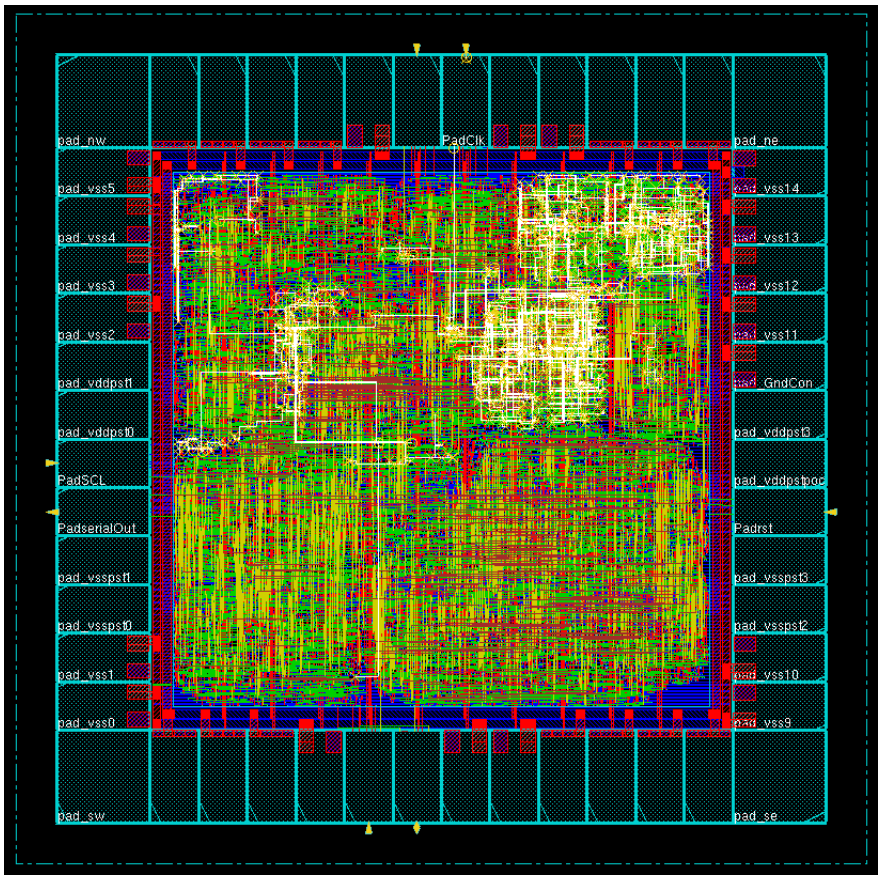

Figure 5.80: Placement with clock tree

5.20.5 Mapping

The figure 5.81 shows the result of the mapping phase, shows the modules distribution of the project modules.

Figure 5.81: Mapping

Results and conclusions

To appropiately develop a noise canceller for audio applications, 16 bits with fixed-point operations are not enough, neverthless the noise canceller works for sine filtering. An adaptive filter is not trivial to implement for dynamic signals because the stability is closely attached to the power of the signal and it has to be compensated with the sensitivity constants (μ and β), this means that they would need to change for each song or even in between different sections of the song; for this reason the *180 degrees phase* method (being used nowadays in most of the active noise cancelling headphones) would be a more robust alternative.

To ensure that the silicon implementation works in the first try, it is really important to have a reference model as reliable as it can be and also a deep verification of each module in the FPGA; by emulating the system the error probability reduces for each model alone and also as a whole system.

This project could be really useful as a basic module in all audio systems. A 16 bit configurable/adaptive filter removes computational load that an equalizer requires helping the development of more complex audio systems.

6.21 Succeses

- The reference model demonstrated the regular FIR filtering and adaptive filtering functionalities with both sines and a song test.
- The flow implementation uses only the I2S bus signals.
- The design was verified comparing the *Modelsim* simulations with the *Signal tap* captures and a functional verification to test the audio output.
- The predefined FIR filters give a clean output.
- The system output and the adaptive filtering sensitivity constants can be configured through I2C bus.
- The adaptive filter works on sine waves as in the reference model.
- A I2C master device was succesfully implemented in a development board to test all the modes.
- There were no problems in the LEC phase in the VLSI, this increases the equivalence from the silicon and FPGA implementations.

- Despite the stability problem, the issue was confirmed in the reference model by limiting the new coefficients to 16 bits.

6.22 Failures/Difficulties

- The adaptive noise cancelling applied to a song didn't work as expected because the word length is fixed to 16 bits, limiting the resolution of the new coefficients calculation; when the power of the signal changes abruptly the stability is lost.

6.23 Potential improvements

- PLL design to convert it into a 64-order filter with the I2S bus without modifying the I2S flow control.
- Hash tables to make the coefficient memories more efficient.
- Convert the design into 24 bit to fix the stability problem in the adaptive filtering mode.

References

(1) Proakis, J. G., & Manolakis, D. G. (1996). *Digital signal processing.* New Jersey: Prentice-Hall, Inc.

(2) Sahu, K., & Sinha, R. (14 de Noviembre de 2018). *Normalized least mean square (NLMS) adaptive filter for noise cancellation.* Obtained from: Technical research organisation India:
http://troindia.in/journal/IJPEMSH/IJPEMSH/paper6.pdf

(3) Sharma, L., & Mehra, R. (14 de Noviembre de 2018). *International Journal of Engineering Trends and Technology.* Obtained from: Adaptive Noise Cancellation using Modified Least Mean Square Algorithm:
https://pdfs.semanticscholar.org/c082/b0390dd13084cad4db7b8b0476d1f564b94c.pdf

(4) Smith, S. W. (1997). *The Scientist and Engineer's Guide to Digital Signal Processing.*

(5) Winder, S. (2002). *Analog and digital filter design.* Woburn, MA: Newnes.

Annexes

8.24 Reference model

8.24.1 Noise cancellation library

noiseCancellation.py

```
from numpy import dot
import numpy as np
from itertools import izip


# Vector sum
def vectorSum(K, L, floatFormat=False):

if floatFormat is False:
return map(sum, izip(K, L))
else:
return map(sum, izip(float(K), float(L)))


# Filter function
def filter(signal, noise, u, N=16):

# if isinstance(signal, list) is True:
# signal = [x/2 for x in vectorSum(signal[0], signal[1])]

# Temporal sample noise vector
noiseS = [0]*N
noiseS[1:N] = noise[0:N-1]
noiseS = list(reversed(noiseS))
# print len(noiseS)
# Filter coefficients
w = [0]*N
w[0] = 0.0

signalOut = []

maxN = []
minN = float(2**8)
```

```
# Filter all signal
for s in range(N, len(signal)):

noiseS[1:N] = noiseS[0:N-1]
noiseS[0] = noise[s-1]
y = dot(noiseS, w)
error = signal[s] - y

# Out signal
signalOut.append(error)
# New coefficients
xpow = np.dot(noiseS, noiseS)
# xpow = 1
w = vectorSum(w, [u*error*x/xpow for x in noiseS])
maxN.append(w)
minN = min(w) if min(w) ¡ minN else minN

return signalOut, maxN, minN
```

8.24.2 Adaptive sine filtering

adaptiveFilterSines.py

```
import scipy.io.wavfile as wav
import numpy as np
from numpy import dot
from itertools import izip
from math import pi, sin
import matplotlib.pyplot as plt


# Vector sum
def vectorSum(K, L, floatFormat=False):

if floatFormat is False:
return map(sum, izip(K, L))
else:
return map(sum, izip(float(K), float(L)))


f1 = 23
f2 = 43
f3 = 13
f4 = 101

sf = f2*20
```

```
signal1 = []
signal2 = []
signal3 = []
signal4 = []
t = []

time = 5

for i in range(0, 5*sf):
signal1.append(sin(2.0*pi*f1*i/sf))
signal2.append(sin(2.0*pi*f2*i/sf))
signal3.append(sin(2.0*pi*f3*i/sf))
signal4.append(sin(2.0*pi*f4*i/sf))
t.append(float(i)/sf)

plotRange = 40

# whiteNoise = np.random.normal(size=len(signal1))
# whiteNoise = [x/(max(whiteNoise)*1.5) for x in whiteNoise]

noise = [x/2 for x in vectorSum(signal2, signal3)]
noise = [x/2 for x in vectorSum(noise, signal4)]

# signal = [x/2 for x in vectorSum(signal1, whiteNoise)]
N = 16
u = .015

# Working with this signals
signal = [x/2 for x in vectorSum(signal1, noise)]

# print signalS, signal[0:N]

noiseS = [0]*N
noiseS[1:N] = noise[0:N-1]
noiseS = list(reversed(noiseS))

w = [0]*N
w[0] = 1.0

signalOut = []

for s in range(N, len(signal)):

noiseS[1:N] = noiseS[0:N-1]
noiseS[0] = noise[s-1]
y = dot(noiseS, w)
error = signal[s] - y
```

```
# if s ¡ 20:
# print "S: ", s
# print "Noise vector: ", noiseS
# print "W: ", w
# print "y: ", y
# print "Song(s): ", signal[s]
# print "Error: ", error

signalOut.append(2*error)

w = vectorSum(w, [u*error*x for x in noiseS])

# plotRange = 200
# initialRange = N + 500
# print len(signalOut), len(signal)

# plt.plot(t[initialRange:plotRange+initialRange], signalOut[initialRange-N:plotRange
+ initialRange - N], 'b')
# plt.plot(t[initialRange:plotRange+initialRange], signal1[initialRange:plotRange+initialRange],
'r')
# plt.plot(t[initialRange:plotRange+initialRange], signal[initialRange:plotRange+initialRange],
'g')
# plt.show()
#
# print len(signal), len(signalOut.extend([0.0]*N))
```

8.24.3 Adaptive song filtering

filterSong.py

```
from numpy import dot
import numpy as np
from itertools import izip


# Vector sum
def vectorSum(K, L, floatFormat=False):

if floatFormat is False:
return map(sum, izip(K, L))
else:
return map(sum, izip(float(K), float(L)))


# Filter function
def filter(signal, noise, u, N=16):
```

```
# if isinstance(signal, list) is True:
# signal = [x/2 for x in vectorSum(signal[0], signal[1])]

# Temporal sample noise vector
noiseS = [0]*N
noiseS[1:N] = noise[0:N-1]
noiseS = list(reversed(noiseS))
# print len(noiseS)
# Filter coefficients
w = [0]*N
w[0] = 0.0

signalOut = []

maxN = []
minN = float(2**8)

# Filter all signal
for s in range(N, len(signal)):

noiseS[1:N] = noiseS[0:N-1]
noiseS[0] = noise[s-1]
y = dot(noiseS, w)
error = signal[s] - y

# Out signal
signalOut.append(error)
# New coefficients
xpow = np.dot(noiseS, noiseS)
# xpow = 1
w = vectorSum(w, [u*error*x/xpow for x in noiseS])
maxN.append(w)
minN = min(w) if min(w) ¡ minN else minN

return signalOut, maxN, minN
```

8.25 Filters coefficient generator

8.25.1 Lowpass coefficient generator script

lowpass_coefficient_generator.py

```
import scipy.signal as signal
import matplotlib.pyplot as plt
import numpy as np
```

```
def a2(value, bits=16):

if value ¡ 0:
value = abs(abs(value) - (1 ¡¡ bits))

return value


def float2binary(num, bits=16, integerPart=1):

value = bin(a2(int(num*(2**(bits-integerPart)))))[2:]
numberOfZeros = bits - len(value)
string = "0"*(bits - len(value))
return string + value


# Parameters
N = 16
integers = 1
sampling_rate = 44.1e3
cut_frequency = 4000
cutoff = 2.0*cut_frequency/sampling_rate


a = signal.firwin(N, cutoff=cutoff, window="hamming")
print np.linalg.norm(a)
b = [float2binary(number, integerPart=integers) for number in a]

print a


fileName = "lowpass_" + str(cut_frequency) + "_n_" + str(N) + ".txt"
print fileName

f = open(fileName, 'w')

for i in b:
# print i
f.write(i + '
n')

f.close()
```

8.25.2 Bandpass coefficient generator script

bandpass_coefficient_generator.py

```
import scipy.signal as signal
import matplotlib.pyplot as plt
import numpy as np


def a2(value, bits=16):

if value ¡ 0:
value = abs(abs(value) - (1 ¡¡ bits))

return value


def float2binary(num, bits=16, integerPart=1):

value = bin(a2(int(num*(2**(bits-integerPart)))))[2:]
numberOfZeros = bits - len(value)
string = "0"*(bits - len(value))
return string + value


# Parameters
N = 16
integers = 1
sampling_rate = 44.1e3
cut_low = 4000
cut_high = 10e3
cutoff_low = 2.0*cut_low/sampling_rate
cutoff_high = 2.0*cut_high/sampling_rate


# a = signal.firwin(N, cutoff=cutoff, window="hamming")
a = signal.firwin(N, [cutoff_low, cutoff_high], pass_zero=False, window="hamming")

print np.linalg.norm(a)
b = [float2binary(number, integerPart=integers) for number in a]

print a


fileName = "bandpass_" + str(int(cut_low)) + "_" + str(int(cut_high)) + "_n_"
+ str(N) + ".txt"
print fileName

f = open(fileName, 'w')
```

```
for i in b:
# print i
f.write(i + '
n')

f.close()
```

8.26 Processor Expert code

```
/* # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # #
# # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # #
# # # # # #
** Filename : main.c
** Project : i2c_slave_adaptive_filter
** Processor : MKL25Z128VLK4
** Version : Driver 01.01
** Compiler : GNU C Compiler
** Date/Time : 2019-06-26, 21:25, # CodeGen: 0
** Abstract :
** Main module.
** This module contains user's application code.
** Settings :
** Contents :
** No public methods
**
** # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # #
# # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # #
# # # # # #*/
/*!
** @file main.c
** @version 01.01
** @brief
** Main module.
** This module contains user's application code.
*/
/*!
** @addtogroup main_module main module documentation
** @
*/
/* MODULE main */


/* Including needed modules to compile this module/procedure */
#include "Cpu.h"
#include "Events.h"
#include "CI2C1.h"
#include "RED.h"
```

```
/* Including shared modules, which are used for whole project */
#include "PE_Types.h"
#include "PE_Error.h"
#include "PE_Const.h"
#include "IO_Map.h"
/* User includes ( #include below this line is not maintained by Processor Expert)
*/
#define true 1
#define false 0
LDD_TError Error;
LDD_TDeviceData *MyI2CPtr;
uint8_t OutData[2] = 0x0, 0x2B;
uint8_t OutData2[5] = 0x03, 0x03, 0x04, 0x5, 0x6;
volatile bool DataTransmittedFlg = FALSE;
volatile bool DataReceivedFlg = FALSE;
uint8_t InpData[16];

LDD_TError I2C_sendBlock(uint8_t *array, uint8_t number)

LDD_TError returnError;

returnError = CI2C1_MasterSendBlock(MyI2CPtr, array, number, LDD_I2C_SEND_STOP);

while (!DataTransmittedFlg)


DataTransmittedFlg = FALSE;

return returnError;



LDD_TError setLowPass1(bool side)

uint8_t OutDataT[2] = 0x0, 0x2;

if (side == true)
OutDataT[1] = OutDataT[1] — 0x40;

LDD_TError returnError;

returnError = I2C_sendBlock(OutDataT, 2);

return returnError;



LDD_TError setLowPass2(void)
```

```
uint8_t OutDataT[2] = 0x0, 0x6;
LDD_TError returnError;

returnError = I2C_sendBlock(OutDataT, 2);

return returnError;


LDD_TError setBandPass1(bool side)

uint8_t OutDataT[2] = 0x0, 0xA;

if (side == true)
OutDataT[1] = OutDataT[1] — 0x40;

LDD_TError returnError;

returnError = I2C_sendBlock(OutDataT, 2);

return returnError;


LDD_TError setBandPass2(void)

uint8_t OutDataT[2] = 0x0, 0xE;
LDD_TError returnError;

returnError = I2C_sendBlock(OutDataT, 2);

return returnError;


LDD_TError setUserDefinedFilter1(void)

uint8_t OutDataT[2] = 0x0, 0x12;
LDD_TError returnError;

returnError = I2C_sendBlock(OutDataT, 2);

return returnError;


LDD_TError setUserDefinedFilter2(void)
```

```
uint8_t OutDataT[2] = 0x0, 0x16;
LDD_TError returnError;

returnError = I2C_sendBlock(OutDataT, 2);

return returnError;


LDD_TError setLMS(void)

uint8_t OutDataT[2] = 0x0, 0x1A;
LDD_TError returnError;

returnError = I2C_sendBlock(OutDataT, 2);

return returnError;


LDD_TError setNLMS(void)

uint8_t OutDataT[2] = 0x0, 0x3A;
LDD_TError returnError;

returnError = I2C_sendBlock(OutDataT, 2);

return returnError;


LDD_TError setBypass(void)

uint8_t OutDataT[2] = 0x0, 0x1;
LDD_TError returnError;

returnError = I2C_sendBlock(OutDataT, 2);

return returnError;


LDD_TError setOff(void)

uint8_t OutDataT[2] = 0x0, 0x0;
LDD_TError returnError;
```

```
returnError = I2C_sendBlock(OutDataT, 2);

return returnError;



LDD_TError writeSDC(uint16_t value)

uint8_t SDC_HP = (uint8_t)(value ¿¿ 8);
uint8_t SDC_LP = (uint8_t)(value & 0xFF);
uint8_t SDC[3] = 0x4, SDC_LP, SDC_HP;

LDD_TError returnError;

returnError = I2C_sendBlock(SDC, 3);

return returnError;



LDD_TError writeSNC(uint16_t value)

uint8_t SNC_HP = (uint8_t)(value ¿¿ 8);
uint8_t SNC_LP = (uint8_t)(value & 0xFF);
uint8_t SNC[3] = 0x2, SNC_LP, SNC_HP;

LDD_TError returnError;

returnError = I2C_sendBlock(SNC, 3);

return returnError;



/*lint -save -e970 Disable MISRA rule (6.3) checking. */
int main(void)
/*lint -restore Enable MISRA rule (6.3) checking. */

/* Write your local variable definition here */

/*** Processor Expert internal initialization.  DON'T REMOVE THIS CODE!!!
***/
PE_low_level_init();
/*** End of Processor Expert internal initialization. ***/

/* Write your code here */
/* For example: for(;;)   */
MyI2CPtr = CI2C1_Init(NULL);
```

```
RED_SetVal(1);

//Error = setBypass();
//Error = writeSNC(0x3fff);
//Error = setLMS();
//Error = setBypass();
//Error = writeSDC(0x1fff);
//Error = writeSNC(0x4000);
//Error = setNLMS();
//Error = setBypass();
Error = setLowPass1(false);
Error = setLowPass1(true);
Error = setBandPass1(false);
Error = setBandPass1(true);
Error = setLowPass2();
Error = setBandPass2();
Error = setBypass();
Error = setOff();

Error = setBypass();

/*** Don't write any code pass this line, or it will be deleted during code generation. ***/
/*** RTOS startup code. Macro PEX_RTOS_START is defined by the RTOS component. DON'T MODIFY THIS CODE!!! ***/
#ifdef PEX_RTOS_START
PEX_RTOS_START(); /* Startup of the selected RTOS. Macro is defined by the RTOS component. */
#endif
/*** End of RTOS startup code. ***/
/*** Processor Expert end of main routine. DON'T MODIFY THIS CODE!!! ***/
for(;;)
/*** Processor Expert end of main routine. DON'T WRITE CODE BELOW!!! ***/
/*** End of main routine. DO NOT MODIFY THIS TEXT!!! ***/

/* END main */
/*!
** @
*/
/*
** # # # # # # # # # # # # # # # # # # # # # # # # # # # # #
# # # # # # # # # # # # # # # # # # # # # # # # # # # # # # #
# # # # # #
**
** This file was created by Processor Expert 10.5 [05.21]
```

```
** for the Freescale Kinetis series of microcontrollers.
**
** # # # # # # # # # # # # # # # # # # # # # # # # # # # # # #
# # # # # # # # # # # # # # # # # # # # # # # # # # # # # # #
# # # # # #
*/
```

8.27 Verilog files

8.27.1 serialFilter (Top)

```
module serialFilter
#
(

parameter WL = 16,
parameter N = 16

)
(

//Input ports
input clk,
input rst,
input serialIn,
input sideSelector,
inout SDA,
input SCL,

//Output ports
output serialOut

);


// ---------------- Wires ------------------
wire [WL - 1 : 0] dataOutL_wire;
wire [WL - 1 : 0] dataOutR_wire;
wire [WL*N - 1 : 0] data_muxL_wire;
wire [WL*N - 1 : 0] data_muxR_wire;
//wire [WL - 1 : 0] data_romR_wire [N - 1 : 0];
//wire [WL - 1 : 0] data_romL_wire [N - 1 : 0];
wire [WL - 1 : 0] MACoutL_wire;
wire [WL - 1 : 0] MACoutR_wire;
wire [WL - 1 : 0] right_data_mux_wire;
wire [WL - 1 : 0] MACoutRegister_wire;
wire [WL - 1 : 0] LeftData_mux_out_wire;
```

```
wire [WL - 1 : 0] RightData_mux_out_wire;
wire [WL - 1 : 0] coefficient_out_wire;
wire [WL - 1 : 0] new_coefficient_out_wire;
wire ADCR_wire;
wire ADCL_wire;
wire repeated_ADCL_wire;
wire data_power_wire;
// Counter outs
wire [$clog2(WL)-1 : 0] leftMAC_counter_out_wire;
wire [$clog2(WL)-1 : 0] rightMAC_counter_out_wire;
wire [$clog2(WL)-1 : 0] coefficient_counter_out_wire;

wire [WL - 1 : 0] bandpass1_coefficient_wire;
wire [WL - 1 : 0] lowpass1_coefficient_wire;
wire [WL - 1 : 0] bandpass2_coefficient_wire;
wire [WL - 1 : 0] lowpass2_coefficient_wire;
wire [WL - 1 : 0] userDefined1_coefficient_wire;
wire [WL - 1 : 0] userDefined2_coefficient_wire;
wire [WL - 1 : 0] adaptive_coefficient_wire;
wire [WL - 1 : 0] shiftRegister_mem_wire;
wire [WL - 1 : 0] x_power_wire;
wire [WL - 1 : 0] LeftMAC_out_register_wire;
wire [WL - 1 : 0] RightMAC_out_register_wire;
wire [WL - 1 : 0] error_wire;
wire [WL - 1 : 0] left_serializer_in_wire;
wire right_serializer_out_wire;
wire adaptive_filter_selected;
wire filtered_out_wire;
wire sideSelector_wire;

// Side selector
wire rightSide;
wire leftSide;
wire LSync_wire;
wire RSync_wire;


// I2C wires
wire [1 : 0] OUTC;
wire [2 : 0] FOPT;
wire AFM;
wire CS;
wire [2*WL - 1 : 0] SNC;
wire [2*WL - 1 : 0] SDC;
wire [2*N*WL - 1 : 0] UDF0;
wire [2*N*WL - 1 : 0] UDF1;

assign rightSide =  sideSelector_wire;
```

```
assign leftSide = sideSelector_wire;

assign sideSelector_wire = (CS == 1'b0)? sideSelector :  sideSelector;



//Adaptive filter sensitivity
localparam u = 16'b0111111111111111;

//Left decoder
LRDecoder
#
(

.WL(1),
.N(WL)

)
LeftDecoder
(

.clk(clk),
.rst(rst),
.serialIn(serialIn),
.sideSelector(leftSide),
.parallelOutput(dataOutL_wire)

);


//Right decoder
LRDecoder
#
(

.WL(1),
.N(WL)

)
RightDecoder
(

.clk(clk),
.rst(rst),
.serialIn(serialIn),
.sideSelector(rightSide),
.parallelOutput(dataOutR_wire)
```

```
);


//Right data register
register
#
(

.WL(WL)

)
RightDataMux
(

.data(dataOutR_wire),
.clk(clk),
.rst(rst),
.enable(leftSide),
.sync_rst(1'b0),
.data_out(right_data_mux_wire)

);


// ********************** I2C interface ***************************

adaptiveFilter_i2c_controller
#(

.WL(8),
.N(N)

)
I2C_controller_with_RegFile
(

.SDA(SDA),
.SCL(SCL),
.clk(clk),
.rst(rst),
.OUTC(OUTC),
.FOPT(FOPT),
.AFM(AFM),
.CS(CS),
.SNC(SNC),
.SDC(SDC),
.UDF0(UDF0),
.UDF1(UDF1)
```

```
);


// ************************* One-shots ***************************

//LR synchronizer
OneShot
#
(

.WORD(1)

)
LSync
(

.iClk(clk),
.iRst(rst),
.iSignal(rightSide),
.oOneShot(LSync_wire)

);


//LR synchronizer
OneShot
#
(

.WORD(1)

)
RSync
(

.iClk(clk),
.iRst(rst),
.iSignal(leftSide),
.oOneShot(RSync_wire)

);


// ************************ End of one-shots ******************************


// ************************** Register chains ***************************
```

```
//Left register MAC chain
registerChain
#
(

.WL(WL),
.N(N)

)
LeftMACDecoder
(

.clk(LSync_wire),
.rst(rst),
.data(dataOutL_wire),
.enable(1'b1),
.dataOut(data_muxL_wire)

);

//Right register MAC chain
registerChain
#
(

.WL(WL),
.N(N)

)
RightMACDecoder
(

.clk(RSync_wire),
.rst(rst),
.data(dataOutR_wire),
.enable(1'b1),
.dataOut(data_muxR_wire)

);


mux_16to1_16_bits
LeftDataMultiplexer
(

.data(data_muxL_wire),
.selector(leftMAC_counter_out_wire),
```

```
.dataOut(LeftData_mux_out_wire)

);


mux_16to1_16_bits
RightDataMultiplexer
(

.data(data_muxR_wire),
.selector(rightMAC_counter_out_wire),
.dataOut(RightData_mux_out_wire)

);


// ************************ End of register chains ******************************


// *************************** MACs ***************************

// Filter mac
serialMAC
#
(

.WL(WL)

)
LeftDataMac
(

.A(LeftData_mux_out_wire),
.B(coefficient_out_wire),
.clk(clk),
.rst(rst),
.enable(leftSide),
.result(MACoutL_wire)

);


// Filter mac
serialMAC
#
(

.WL(WL)
```

```
)
RightDataMac
(

.A(RightData_mux_out_wire),
.B(coefficient_out_wire),
.clk(clk),
.rst(rst),
.enable(rightSide),
.result(MACoutR_wire)

);

// Data power mac
serialMAC
#
(

.WL(WL)

)
PowerMAC
(

.A(LeftData_mux_out_wire),
.B(LeftData_mux_out_wire),
.clk(clk),
.rst(rst),
.enable(leftSide),
.result(x_power_wire)

);


// ************************ End of MACs ******************************


// ************************ Memories *********************************


// Lowpass filter coefficients
serialRom_lowpass1
#
(

.DATA_WIDTH(WL),
.ADDR_WIDTH($clog2(WL))
```

```
)
LowpassCoefficients1
(

.addr(coefficient_counter_out_wire),
.q(lowpass1_coefficient_wire)

);

// Lowpass filter coefficients
serialRom_lowpass2
#
(

.DATA_WIDTH(WL),
.ADDR_WIDTH($clog2(WL))

)
LowpassCoefficients2
(

.addr(coefficient_counter_out_wire),
.q(lowpass2_coefficient_wire)

);


// Bandpass filter coefficients
serialRom_bandpass1
#
(

.DATA_WIDTH(WL),
.ADDR_WIDTH($clog2(WL))

)
BandpassCoefficients1
(

.addr(coefficient_counter_out_wire),
.q(bandpass1_coefficient_wire)

);

// Bandpass filter coefficients
serialRom_bandpass2
#
```

```
(

.DATA_WIDTH(WL),
.ADDR_WIDTH($clog2(WL))

)
BandpassCoefficients2
(

.addr(coefficient_counter_out_wire),
.q(bandpass2_coefficient_wire)

);


// Adaptive filter coefficients (Rewritable)
simple_dual_port_ram_single_clock
#
(

.DATA_WIDTH(WL),
.ADDR_WIDTH($clog2(WL))

)
AdaptativeFilterRAM
(

.data(new_coefficient_out_wire),
.read_addr(coefficient_counter_out_wire),
.write_addr(rightMAC_counter_out_wire),
.we(rightSide),
.clk(clk),
.rst(rst),
.q(adaptive_coefficient_wire)

);


//Shift register data memory (Save data shift register) (Rewritable)
simple_dual_port_ram_single_clock
#
(

.DATA_WIDTH(WL),
.ADDR_WIDTH($clog2(WL))

)
shiftRegisterDataMemory
```

```
(

.data(LeftData_mux_out_wire),
.read_addr(rightMAC_counter_out_wire),
.write_addr(leftMAC_counter_out_wire),
.we(leftSide),
.clk(clk),
.rst(rst),
.q(shiftRegister_mem_wire)

);


mux_16to1_16_bits
UserDefinedFilter1_MUX
(

.data(UDF0),
.selector(coefficient_counter_out_wire),
.dataOut(userDefined1_coefficient_wire)

);


mux_16to1_16_bits
UserDefinedFilter2_MUX
(

.data(UDF1),
.selector(coefficient_counter_out_wire),
.dataOut(userDefined2_coefficient_wire)

);


// ********************* End of memories ****************************


//Main MAC output reigster
register
#
(

.WL(WL)

)
LeftMAC_out_register
(
```

```
.data(MACoutL_wire),
.clk(clk),
.rst(rst),
.enable(leftSide),
.sync_rst(1'b0),
.data_out(LeftMAC_out_register_wire)

);


mux_8to1_16_bits
CoefficientMux
(

.data(16'b0, adaptive_coefficient_wire, userDefined2_coefficient_wire, userDefined1_coefficient_wire,
bandpass2_coefficient_wire, bandpass1_coefficient_wire, lowpass2_coefficient_wire, low-
pass1_coefficient_wire),
.selector(FOPT),
.dataOut(coefficient_out_wire)

);


// ********************* Counters ****************************


// Address generator
counter
#
(
.COUNT(WL)
)
MAC_address_generator
(
.clk(clk),
.rst(rst),
.sync_rst(rightSide),
.count(leftSide),
.countOut(leftMAC_counter_out_wire)
);

//Coefficient address generator
counter
#
(
.COUNT(WL)
)
```

```
coeff_address_generator
(
.clk(clk),
.rst(rst),
.sync_rst(leftSide),
.count(rightSide),
.countOut(rightMAC_counter_out_wire)
);


assign coefficient_counter_out_wire = leftSide? leftMAC_counter_out_wire : right-
MAC_counter_out_wire;


// ********************* End of counters ****************************


//New coefficient calculator
newCoefficient
#
(

.WL(WL)

)
NewCoefficientGenerator
(

.x_in(shiftRegister_mem_wire),
.x_power(x_power_wire),
.error(error_wire),
.SDC(SDC),
.SNC(SNC),
.AFM(AFM),
.w(adaptive_coefficient_wire),
.result(new_coefficient_out_wire)

);


// ********************* Serializers ****************************


//Left shift register
shift_Register
#
(
```

```
.WL(WL)

)
leftSerializer
(

.data_to_transmit(left_serializer_in_wire),
.enable(leftSide),
.reset(rst),
.clk(clk),
.shift(rightSide),
.Serial_Output(ADCL_wire)

);


//Right shift register
shift_Register
#
(

.WL(WL)

)
rightSerializer
(

.data_to_transmit(MACoutR_wire),
.enable(rightSide),
.reset(rst),
.clk(clk),
.shift(leftSide),
.Serial_Output(ADCR_wire)

);

// Left channel repeater
serial_delay
#
(
.delay(WL)
)
leftChannelRepeater
(

.clk(clk),
.signal(ADCL_wire),
.rst(rst),
```

```
.delayed_signal(repeated_ADCL_wire)

);



// ********************* End of serializers ****************************


mux_2to1_1_bits
OutputMux
(

.data(ADCL_wire, right_serializer_out_wire),
.selector(rightSide),
.dataOut(filtered_out_wire)

);


mux_3to1_1_bits
OUTC_mux
(

.data(1'b0, filtered_out_wire, serialIn, 1'b0),
.selector(OUTC),
.dataOut(serialOut)

);


assign left_serializer_in_wire = (adaptive_filter_selected)? error_wire : MACoutL_wire;

assign right_serializer_out_wire = (adaptive_filter_selected)? repeated_ADCL_wire
: ADCR_wire;

assign error_wire = right_data_mux_wire - LeftMAC_out_register_wire;

assign adaptive_filter_selected = (FOPT == 3'b110)? 1'b1 : 1'b0;

endmodule
```

8.27.2 adaptiveFilter_i2c_controller

```
module adaptiveFilter_i2c_controller
#(
```

```
parameter WL = 8,
parameter N = 16

)(

inout SDA,
input SCL,
input rst,
input clk,

output [1 : 0] OUTC,
output [2 : 0] FOPT,
output AFM,
output CS,
output [2*WL - 1 : 0] SNC,
output [2*WL - 1 : 0] SDC,
output [2*N*WL - 1 : 0] UDF0,
output [2*N*WL - 1 : 0] UDF1

);

// ———————— Wires ————————


//I2C receiver wires
wire [WL - 1 : 0] i2c_data_wire;
wire i2c_dataReady_wire;
wire [6 : 0] i2c_address_wire;
wire i2c_start_wire;
//I2C state machine wires
wire [WL - 1 : 0] stateMachine_out_wire;
wire [WL - 1 : 0] stateMachine_address_wire;
wire stateMachine_we_wire;
wire SCL_wire;

register
#(

.WL(1)

)
Sync
(

.clk(clk),
.rst(rst),
.data(SCL),
```

```
.sync_rst(1'b0),
.enable(1'b1),
.data_out(SCL_wire)

);



i2c_slave
I2CReceiver
(

.SCL(SCL_wire),
.SDA(SDA),
.rst(rst),
.I2C_ADR(i2c_address_wire),
.IOout(i2c_data_wire),
.dataReady(i2c_dataReady_wire),
.start(i2c_start_wire)

);


adaptiveFilter_state_machine
#(

.WL(WL)

)
FSM
(

.clk(SCL),
.rst(rst),
.in(i2c_dataReady_wire),
.start(i2c_start_wire),
.dataIn(i2c_data_wire),
.out(stateMachine_out_wire),
.address(stateMachine_address_wire),
.we(stateMachine_we_wire)

);


registerFile
#(

.N(N)
```

```
)
MainRegisterFile
(

.clk(SCL),
.rst(rst),
.in_data(i2c_data_wire),
.address(stateMachine_address_wire),
.we(stateMachine_we_wire),
.OUTC(OUTC),
.FOPT(FOPT),
.AFM(AFM),
.CS(CS),
.I2C(i2c_address_wire),
.SNC(SNC),
.SDC(SDC),
.UDF0(UDF0),
.UDF1(UDF1)

);



endmodule
```

8.27.3 adaptiveFilter_i2c_state_machine

```
module adaptiveFilter_state_machine
#(

parameter WL = 8

)(

input clk, in, start, rst,
input [WL - 1 : 0] dataIn,
output [WL - 1 : 0] out, address,
output reg we

);

reg [WL - 1 : 0] address_reg;
reg [1 : 0] state;

localparam IDLE=0, WAIT_ADDRESS=1, DATA_RECEIVED=2, WRITE_ENABLE=3;
```

```
always @(state)
begin
case (state)
IDLE:
we = 1'b0;
WAIT_ADDRESS:
we = 1'b0;
DATA_RECEIVED:
we = 1'b0;
WRITE_ENABLE:
we = 1'b1;
default:
we = 1'b0;
endcase
end

always @(posedge clk or negedge rst or negedge start)
begin
if (!rst — !start) begin
state = IDLE;
address_reg = (WL)1'b0;
end
else
case (state)

IDLE:

if(start) begin
state = WAIT_ADDRESS;
end
else
state = IDLE;

WAIT_ADDRESS:

if (in) begin
state = DATA_RECEIVED;
address_reg = dataIn;
end

else if(!start)
state = IDLE;

else
state = WAIT_ADDRESS;

DATA_RECEIVED:
```

```
if (in) begin
state = WRITE_ENABLE;
end

else if(!start)
state = IDLE;

else
state = DATA_RECEIVED;

WRITE_ENABLE:

if(!start)
state = IDLE;

else begin

address_reg = address_reg + (WL-1)1'b0, 1'b1;
state = DATA_RECEIVED;

end


endcase
end


assign address = address_reg;
assign out = dataIn;


endmodule
```

8.27.4 counter

```
module counter
#
(

parameter COUNT = 16,
parameter WL = $clog2(COUNT)

)
(

input clk,
input rst,
```

```
input count,
input sync_rst,

output reg [WL - 1 : 0] countOut

);


always @(posedge clk, negedge rst) begin

if(!rst)
countOut = WL1'b0;

else if(count)
countOut = countOut + 1'b1;

else if(sync_rst)
countOut = WL1'b0;

end



endmodule
```

8.27.5 division

```
module division
#(
parameter Word_Length = 6,
parameter Integer_Part = 3,
parameter Fractional_Part = Word_Length - Integer_Part
)
(
// Input Ports
input signed [Word_Length-1:0] A,B,

// Output Ports
output signed[Word_Length-1:0] D
);

wire signed [2*Word_Length-1:0] X_wire;
wire signed [Word_Length-1:0] X_Trunc_wire, D_wire;



assign X_wire = (A ¡¡ (Word_Length-Integer_Part)) / B;
```

```
assign D = X_wire[Word_Length-1:0];

endmodule
```

8.27.6 Fixed_Point_MAC

```
module Fixed_Point_MAC
#(
parameter Word_Length = 6,
parameter Integer_Part = 3,
parameter Fractional_Part = Word_Length - Integer_Part
)
(
// Input Ports
input signed [Word_Length-1:0] A,B,C,

// Output Ports
output signed [Word_Length-1:0] D
);

wire signed [2*Word_Length-1:0] X_wire;
wire signed [Word_Length-1:0] X_Trunc_wire, D_wire;



assign X_wire = A * B;
assign D = X_wire[2*Word_Length-1-Integer_Part:Fractional_Part] + C;

endmodule
```

8.27.7 i2c_slave

8.27.8 LRDecoder

```
module LRDecoder #(
parameter WL = 1,
parameter N = 16
)(
//Input ports
input clk,
input rst,
input serialIn,
input sideSelector,

//Output ports
output [N- 1 : 0] parallelOutput
);
```

```
//Logics
wire [N - 1 : 0] parOut ;

//int i;

//Register chain
registerChain
#
(
.WL(WL),
.N(N)
)
RegisterChain
(
.clk(clk),
.rst(rst),
.data(serialIn),
.enable(sideSelector),
.dataOut(parOut)
);

assign parallelOutput = parOut;


endmodule
```

8.27.9 mux_2to1_1_bits

```
module mux_2to1_1_bits(

input [1 : 0] data,
input [0 : 0] selector,
output reg [0 : 0] dataOut

);

always @(*)
case(selector)
1'b0: dataOut = data[0 : 0];
1'b1: dataOut = data[1 : 1];
endcase

endmodule
```

8.27.10 mux_3to1_1_bits

```
module mux_3to1_1_bits(

input [2 : 0] data,
input [1 : 0] selector,
output reg [0 : 0] dataOut

);

always @(*)
case(selector)
2'b0: dataOut = data[0 : 0];
2'b1: dataOut = data[1 : 1];
2'b10: dataOut = data[2 : 2];
default: dataOut = data[0 : 0];
endcase

endmodule
```

8.27.11 mux_8to1_16_bits

```
module mux_8to1_16_bits(

input [127 : 0] data,
input [2 : 0] selector,
output reg [15 : 0] dataOut

);

always @(*)
case(selector)
3'b0: dataOut = data[15 : 0];
3'b1: dataOut = data[31 : 16];
3'b10: dataOut = data[47 : 32];
3'b11: dataOut = data[63 : 48];
3'b100: dataOut = data[79 : 64];
3'b101: dataOut = data[95 : 80];
3'b110: dataOut = data[111 : 96];
3'b111: dataOut = data[127 : 112];
endcase

endmodule
```

8.27.12 mux_16to1_16_bits

```
module mux_16to1_16_bits(

input [255 : 0] data,
input [3 : 0] selector,
output reg [15 : 0] dataOut

);

always @(*)
case(selector)
4'b0: dataOut = data[15 : 0];
4'b1: dataOut = data[31 : 16];
4'b10: dataOut = data[47 : 32];
4'b11: dataOut = data[63 : 48];
4'b100: dataOut = data[79 : 64];
4'b101: dataOut = data[95 : 80];
4'b110: dataOut = data[111 : 96];
4'b111: dataOut = data[127 : 112];
4'b1000: dataOut = data[143 : 128];
4'b1001: dataOut = data[159 : 144];
4'b1010: dataOut = data[175 : 160];
4'b1011: dataOut = data[191 : 176];
4'b1100: dataOut = data[207 : 192];
4'b1101: dataOut = data[223 : 208];
4'b1110: dataOut = data[239 : 224];
4'b1111: dataOut = data[255 : 240];
endcase

endmodule
```

8.27.13 newCoefficient

```
module newCoefficient
#(
parameter WL = 16,
parameter INT = 1
)
(
//Input ports
input signed [WL-1 : 0] x_in, x_power, error, SDC, SNC, w,
input AFM,

//Output ports
output signed [WL-1 : 0] result
);
```

```
// Wires
wire [WL-1 : 0] error_u_out_w;
wire [WL-1 : 0] x_in_out_w;
wire [WL-1 : 0] variable_u;
wire [WL-1 : 0] NLMS_wire;

wire [WL-1 : 0] minSignal;


// Multiplication unit
Fixed_Point_MAC
#(
.Word_Length(WL),
.Integer_Part(INT)
)
error_u_MAC
(
.A(error),
.B(variable_u),
.C(1'b0),
.D(error_u_out_w)
);


Fixed_Point_MAC
#(
.Word_Length(WL),
.Integer_Part(INT)
)
x_in_MAC
(
.A(error_u_out_w),
.B(x_in),
.C(w),
.D(x_in_out_w)
);


division
#(

.Word_Length(WL),
.Integer_Part(INT)

)
VariableStepSize
(
```

```
.A(SNC),
.B(SDC + (x_power)),
.D(NLMS_wire)

);

assign result = x_in_out_w;
assign variable_u = (AFM == 1)? NLMS_wire : SNC;

endmodule
```

8.27.14 OneShot

```
module OneShot #(parameter WORD = 1)
(
input iClk,
input iRst,
input [WORD-1:0] iSignal,
output [WORD-1:0] oOneShot
);

reg [WORD-1:0]rOneShot;

always@ (posedge iClk, negedge iRst)
begin
if ( iRst)
begin
rOneShot = WORD1'b0;
end
else
begin
rOneShot = iSignal;
end
end

assign oOneShot = rOneShot;

endmodule
```

8.27.15 register

```
module register
#
(
```

```
parameter WL = 4

)
(

input [WL - 1 : 0] data,
input clk,
input rst,
input enable,
input sync_rst,

output reg [WL - 1 : 0] data_out

);


always @(posedge clk, negedge rst) begin //ThisIsARegister

if(!rst)
data_out ¡= 0;

else if(enable) begin //Assignations

if(sync_rst)
data_out ¡= 0;

else
data_out ¡= data;

end //Assignations


end //ThisIsARegister



endmodule
```

8.27.16 register_w_initial_conditions

```
module register_w_initial_conditions
#
(

parameter WL = 4

)
```

```
(

input [WL - 1 : 0] data,
input clk,
input rst,
input enable,
input sync_rst,
input [WL - 1 : 0] initial_condition,

output reg [WL - 1 : 0] data_out

);


always @(posedge clk, negedge rst) begin //ThisIsARegister

if(!rst)
data_out ¡= initial_condition;

else if(enable) begin //Assignations

if(sync_rst)
data_out ¡= 0;

else
data_out ¡= data;

end //Assignations


end //ThisIsARegister



endmodule
```

8.27.17 registerChain

```
module registerChain
#
(

parameter WL = 16,
parameter N = 3

)
(
```

```
//Input ports
input clk,
input rst,
input [WL - 1 : 0] data,
input enable,

//Output ports
output [N*WL-1 : 0] dataOut //[N - 1 : 0]

);

//Wires
wire [(N+1)*WL-1 : 0] dataOut_wire/* [N : 0]*/;

genvar i;

//Register chain generation
generate

for(i = 0; i ¡ N; i = i + 1) begin : Registers

register
#
(
.WL(WL)
)
RegisterModule
(
.data(dataOut_wire[(i+1)*WL - 1 : i*WL]),
.clk(clk),
.rst(rst),
.enable(enable),
.sync_rst(1'b0),
.data_out(dataOut_wire[(i+2)*WL-1:(i+1)*WL])
);

assign dataOut[(i+1)*WL-1:(i)*WL] = dataOut_wire[(i+2)*WL-1:(i+1)*WL];

end

endgenerate

assign dataOut_wire[WL - 1 : 0] = data;

endmodule
```

8.27.18 registerFile

```
module registerFile
#(

parameter N = 16

)
(

// Inputs
input [WL - 1 : 0] in_data,
input [$clog2(ADDR_WIDTH) - 1 : 0] address,
input we,
input clk,
input rst,

//Outputs
output [1 : 0] OUTC,
output [2 : 0] FOPT,
output AFM,
output CS,
output [6:0] I2C,
output [2*WL - 1 : 0] SNC,
output [2*WL - 1 : 0] SDC,
output [2*N*WL - 1 : 0] UDF0,
output [2*N*WL - 1 : 0] UDF1

);


localparam WL = 8;
localparam ADDR_WIDTH = 70;

//Generate general wires
wire [ADDR_WIDTH - 1 : 0] enable_out_wire;
wire [WL*ADDR_WIDTH - 1 : 0] outs_wire;
wire [WL*ADDR_WIDTH - 1 : 0] initials_wire;


//Initial conditions
wire [1 : 0] OUTC_initial;
wire [2 : 0] FOPT_initial;
wire AFM_initial;
wire CS_initial;
wire [1 : 0] MCR_unused_initial;
wire [6:0] I2C_initial;
wire I2C_unused_initial;
wire [2*WL - 1 : 0] SNC_initial;
```

```
wire [2*WL - 1 : 0] SDC_initial;
wire [2*N*WL - 1 : 0] UDF0_initial;
wire [2*N*WL - 1 : 0] UDF1_initial;


//————————Initial conditions assignment————————
assign OUTC_initial = 2'b01; //Default: bypass
assign FOPT_initial = 3'b000; //Default: low pass filter
assign AFM_initial = 1'b0; //Default: LMS
assign CS_initial = 1'b0; //Default : I2S default
assign MCR_unused_initial = 2'b00;
assign I2C_initial = 7'h45; //Default: 0x45
assign I2C_unused_initial = 1'b0;
assign SNC_initial = 2*WL1'b1 ¿¿ 1; //Default: 0
assign SDC_initial = 2*WL1'b1 ¿¿ 1; //Default: 0
assign UDF0_initial = 2*N*WL1'b0; //Default: 0
assign UDF1_initial = 2*N*WL1'b0; //Default: 0


//————————Output wires declaration————————
wire [1 : 0] OUTC_wire;
wire [2 : 0] FOPT_wire;
wire AFM_wire;
wire CS_wire;
wire [6:0] I2C_wire;
wire [2*WL - 1 : 0] SNC_wire;
wire [2*WL - 1 : 0] SDC_wire;
wire [2*N*WL - 1 : 0] UDF0_wire;
wire [2*N*WL - 1 : 0] UDF1_wire;

//————————Output wires assignation————————
assign OUTC_wire = outs_wire[1 : 0];
assign FOPT_wire = outs_wire[4 : 2];
assign AFM_wire = outs_wire[5];
assign CS_wire = outs_wire[6];
assign I2C_wire = outs_wire[14 : 8];
assign SNC_wire = outs_wire[31 : 16];
assign SDC_wire = outs_wire[47 : 32];
assign UDF0_wire = outs_wire[303 : 48];
assign UDF1_wire = outs_wire[559 : 304];

assign OUTC = OUTC_wire;
assign FOPT = FOPT_wire;
assign AFM = AFM_wire;
assign CS = CS_wire;
assign I2C = I2C_wire;
assign SNC = SNC_wire;
assign SDC = SDC_wire;
```

```
assign UDF0 = UDF0_wire;
assign UDF1 = UDF1_wire;


assign initials_wire =  UDF1_initial,
UDF0_initial,
SDC_initial,
SNC_initial,
I2C_unused_initial,
I2C_initial,
MCR_unused_initial,
AFM_initial,
FOPT_initial,
OUTC_initial;



//Address to write enable converter
single_bit_param_demux
#(

.N(ADDR_WIDTH)

)
ADDRESS_CONVERTER
(

.selector(address),
.dataOut(enable_out_wire)

);


genvar i;




// All registers instantiation
generate


for(i=0; i ¡= ADDR_WIDTH - 1; i = i + 1) begin : Registers

register_w_initial_conditions
#(

.WL(WL)
```

```
)
IndividualRegister
(

.clk(clk),
.rst(rst),
.data(in_data),
.enable(enable_out_wire[i] & we),
.sync_rst(1'b0),
.initial_condition(initials_wire[(i+1)*WL - 1 : i*WL]),
.data_out(outs_wire[(i+1)*WL - 1 : i*WL])

);

end

endgenerate


endmodule
```

8.27.19 serial_delay

```
module serial_delay
#(
parameter delay = 8
)
(
// Input Ports
input clk,
input signal,
input rst,

// Output Ports
output delayed_signal
);

wire [delay : 0] register_connector;
genvar i;

generate

for (i=0; i ¡ delay; i=i+1) begin : registerChain
register
#(
.WL(1)
```

```
)
registerChainBlock
(

.clk(clk),
.rst(rst),
.enable(1'b1),
.sync_rst(1'b0),
.data(register_connector[i]),
.data_out(register_connector[i+1])

);

end

endgenerate


assign register_connector[0] = signal;
assign delayed_signal = register_connector[delay];

endmodule
```

8.27.20 serialMAC

```
module serialMAC
#(
parameter WL = 16,
parameter INT = 1
)
(
//Input ports
input [WL-1 : 0] A, B,
input clk, rst, enable,

//Output ports
output [WL-1 : 0] result
);


// Wires
wire [WL-1 : 0] MAC_out_w;
wire [WL-1 : 0] fdbk_reg_w;



// Multiplication unit
```

```
Fixed_Point_MAC
#(
.Word_Length(WL),
.Integer_Part(INT)
)
ALU
(
.A(A),
.B(B),
.C(fdbk_reg_w),
.D(MAC_out_w)
);


// Feedback register
register
#(
.WL(WL)
)
Feedback_register
(
.data(MAC_out_w),
.clk(clk),
.rst(rst),
.enable(1'b1),
.sync_rst( enable),
.data_out(fdbk_reg_w)
);


//// Address generator
//counter
//#
//(
// .COUNT(WL)
//)
//Address_generator
//(
// .clk(clk),
// .rst(rst),
// .sync_rst( enable),
// .count(enable),
// .countOut(address)
//);


assign result = fdbk_reg_w;
```

```
endmodule
```

8.27.21 serialRom_bandpass1

```
// Quartus II Verilog Template
// Single Port ROM

module serialRom_bandpass1
#(parameter DATA_WIDTH=16, parameter ADDR_WIDTH=5)
(
input [(ADDR_WIDTH-1):0] addr,
output [(DATA_WIDTH-1):0] q
);

// Declare the ROM variable
wire [(DATA_WIDTH*(2**ADDR_WIDTH))-1:0] rom;

// Initialize the ROM with $readmemb. Put the memory contents
// in the file single_port_rom_init.txt. Without this file,
// this design will not compile.

assign rom = 256'h47ff7ffed8fd8af5a1fe5bdad33e913e91dad3fe5bf5a1fd8afed8ff7f0047;


mux_16to1_16_bits
BandPassROM
(
.data(rom),
.selector(addr),
.dataOut(q)
);

endmodule
```

8.27.22 serialRom_bandpass2

```
// Quartus II Verilog Template
// Single Port ROM

module serialRom_bandpass2
#(parameter DATA_WIDTH=16, parameter ADDR_WIDTH=5)
(
input [(ADDR_WIDTH-1):0] addr,
output [(DATA_WIDTH-1):0] q
);
```

```
// Declare the ROM variable
wire [(DATA_WIDTH*(2**ADDR_WIDTH))-1:0] rom;

// Initialize the ROM with $readmemb. Put the memory contents
// in the file single_port_rom_init.txt. Without this file,
// this design will not compile.

assign rom = 256'hfffa009c0200fe3df0f8eb750298233523350298eb75f0f8fe3d0200009cfffa;


mux_16to1_16_bits
BandPassROM
(
.data(rom),
.selector(addr),
.dataOut(q)
);

endmodule
```

8.27.23 serialRom_lowpass1

```
// Quartus II Verilog Template
// Single Port ROM

module serialRom_lowpass1
#(parameter DATA_WIDTH=16, parameter ADDR_WIDTH=5)
(
input [(ADDR_WIDTH-1):0] addr,
output [(DATA_WIDTH-1):0] q
);

// Declare the ROM variable
wire [(DATA_WIDTH*(2**ADDR_WIDTH))-1:0] rom;

// Initialize the ROM with $readmemb. Put the memory contents
// in the file single_port_rom_init.txt. Without this file,
// this design will not compile.


assign rom = 256'h13301d20391062b092e0c150e5c0f9c0f9c0e5c0c15092e062b039101d20133;

mux_16to1_16_bits
LowPassROM
(
.data(rom),
```

```
.selector(addr),
.dataOut(q)
);

endmodule
```

8.27.24 serialRom_lowpass2

```
// Quartus II Verilog Template
// Single Port ROM

module serialRom_lowpass2
#(parameter DATA_WIDTH=16, parameter ADDR_WIDTH=5)
(
input [(ADDR_WIDTH-1):0] addr,
output [(DATA_WIDTH-1):0] q
);

// Declare the ROM variable
wire [(DATA_WIDTH*(2**ADDR_WIDTH))-1:0] rom;

// Initialize the ROM with $readmemb. Put the memory contents
// in the file single_port_rom_init.txt. Without this file,
// this design will not compile.


assign rom = 256'hff97ff950003020e06860cfb138b17b517b5138b0cfb0686020e0003ff95ff97;

mux_16to1_16_bits
LowPassROM
(
.data(rom),
.selector(addr),
.dataOut(q)
);

endmodule
```

8.27.25 shift_Register

```
module shift_Register
#(
parameter WL=8
)
(
input [(WL-1):0] data_to_transmit,
```

```
input enable,
input reset,
input clk,
input shift,
output Serial_Output
);

reg [(WL-1):0] Shift_Register_data;

always @( posedge clk, negedge reset ) begin

if(!reset) begin
Shift_Register_data ¡= WL1'b0;
end
else if(enable) begin
Shift_Register_data ¡= data_to_transmit;
end
else if(shift) begin
Shift_Register_data ¡= (Shift_Register_data) ¡¡ (1);
end
else begin
Shift_Register_data ¡= Shift_Register_data ;
end

end// end always

assign Serial_Output = Shift_Register_data[WL - 1];

//////////////////////////////////////////////////////////////


endmodule
```

8.27.26 simple_dual_port_ram_single_clock

```
// Quartus II Verilog Template
// Simple Dual Port RAM with separate read/write addresses and
// single read/write clock

module simple_dual_port_ram_single_clock
#(parameter DATA_WIDTH=8, parameter ADDR_WIDTH=2)
(
input [(DATA_WIDTH-1):0] data,
input [(ADDR_WIDTH-1):0] read_addr, write_addr,
input we, clk, rst,
output [(DATA_WIDTH-1):0] q
);
```

```
wire [2**ADDR_WIDTH - 1 : 0] selector_enable;
wire [(2**ADDR_WIDTH)*DATA_WIDTH - 1 : 0] registers_out;

//Enable selector
single_bit_param_demux
#(
.N(2**ADDR_WIDTH)
)
EnableSelector
(

.selector(write_addr),
.dataOut(selector_enable)

);

// Output selection
mux_16to1_16_bits
OutputMux
(

.data(registers_out),
.selector(read_addr),
.dataOut(q)

);


// ALL REGISTERS INSTANTIATIONS ----------------------------------
genvar i;

//Register chain generation
generate

for(i = 0; i ¡ (2**ADDR_WIDTH); i = i + 1) begin : Registers

register
#
(
.WL(DATA_WIDTH)
)
RegisterModule
(
.data(data),
.clk(clk),
.rst(rst),
```

```
.enable(selector_enable[i]&we),
.sync_rst(1'b0),
.data_out(registers_out[(i+1)*DATA_WIDTH - 1 : i*DATA_WIDTH])
);


end

endgenerate

endmodule
```

8.27.27 single_bit_param_demux

```
module single_bit_param_demux
#(

parameter N = 4

)(

input wire [$clog2(N) - 1 : 0] selector,
output wire [N - 1 : 0] dataOut

);


assign dataOut = (N-1)1'b0,1'b1 ¡¡ (selector);

endmodule
```

8.28 VLSI files

8.28.1 Clock tree synthesis script

```
# ITESO University
# Cuauhtemoc Aguilera
# This script should be sourced affter power grid has been defined and before to
create CTS
#
Puts "Starting to do Design Placement..."
# Flooplan Mode for prototyping, runs quickly
setPlaceMode -fp false
placeDesign
Puts "... finished Design Placement"
```

```
# To report unrouted nets
#set trPrintIgnoredPadNets ¡limit¿

# Use the FE-CTS
setCTSMode -engine ck

# Create clock tree using the clock buffers list:

clockDesign -specFile ../Clock.ctstch -outDir clock_report -fixedInstBeforeCTS
displayClockTree -skew -allLevel -clkRouteOnly

# Edit the .ctstch that was created to complete constraints...
```

8.28.2 LEC

```
# Script to launch Conformal lec

# RTL versus Intermediate
lec -XL -do ./outputs_Jul15-20:43:29/rtl2intermediate.lec.do


# Intermediate versus Map
#lec -XL -do ./outputs_Jul13-19:20:58/intermediate2final.lec.do

# RTL versus Map
#lec -XL -do ./outputs_Jul13-19:20:58/rtl2intermediate.lec.do
```

8.28.3 RC

```
rc -f ../Filter_simple_RC14_26_BiCMOS.tcl -gui -log Filter.log
```

8.28.4 Power grid

```
# ITESO University
# Cuauhtémoc Aguilera


# Defining process mode
setDesignMode -process 130

# Defining floorplan
getIoFlowFlag
setFPlanRowSpacingAndType 3.6 2
setIoFlowFlag 0
setIoFlowFlag 0
floorPlan -site CORE -r 0.774962742176 0.508999 36 36 36 36
```

```
uiSetTool select
getIoFlowFlag
fit

# Defining global nets
clearGlobalNets
globalNetConnect VDD -type pgpin -pin VDD -inst * -module  -verbose
globalNetConnect VSS -type pgpin -pin VSS -inst * -module  -verbose
globalNetConnect VDD -type tiehi -pin VDD -inst * -module  -verbose
globalNetConnect VSS -type tielo -pin VSS -inst * -module  -verbose

# Creating power rins
set sprCreateIeRingNets
set sprCreateIeRingLayers
set sprCreateIeRingWidth 1.0
set sprCreateIeRingSpacing 1.0
set sprCreateIeRingOffset 1.0
set sprCreateIeRingThreshold 1.0
set sprCreateIeRingJogDistance 1.0

addRing -skip_via_on_wire_shape Noshape -skip_via_on_pin Standardcell -center 1 -
stacked_via_top_layer AM -type core_rings -jog_distance 0.2 -threshold 0.2 -nets VDD
VSS -follow core -stacked_via_bottom_layer M1 -layer bottom M1 top M1 right M2
left M2 -width 10 -spacing 5 -offset 0.2


# Adding horizontal stripes
sroute -connect  blockPin padPin padRing corePin floatingStripe  -layerChangeRange
M1 AM  -blockPinTarget  nearestTarget  -padPinPortConnect  allPort oneGeom  -
padPinTarget  nearestTarget  -corePinTarget  firstAfterRowEnd  -floatingStripeTarget
blockring padring ring stripe ringpin blockpin followpin  -allowJogging 1 -crossoverViaLayerRange
M1 AM  -nets  VDD VSS  -allowLayerChange 1 -blockPin useLef -targetViaLayerRange
M1 AM

# Adding vertical stripes

addStripe -skip_via_on_wire_shape Noshape -block_ring_top_layer_limit MQ -max_same_layer_jog_length
4  -padcore_ring_bottom_layer_limit  M1  -number_of_sets  50  -skip_via_on_pin  Stan-
dardcell -stacked_via_top_layer AM -padcore_ring_top_layer_limit MQ -spacing 0.25 -
xleft_offset 15 -xright_offset 15 -merge_stripes_value 0.2 -layer M2 -block_ring_bottom_layer_limit
M1 -width 0.5 -nets VDD VSS -stacked_via_bottom_layer M1
```

8.28.5 Frame

```
# ITESO University
# Cuauhtemoc Aguilera
# This script should be sourced affter the
# .ctstch file has been modified for completing constraints



Puts "Timing the design before CTS"

# Calculates the delays for paths based on max. opertating conditions (op) and min. op.
setAnalysisMode -analysisType onChipVariation

timeDesign -preCTS -prefix preCTS_setup
timeDesign -preCTS -prefix preCTS_hold -hold

Puts "Running CTS"
dbDeleteTrialRoute
clockDesign -specFile ../Clock.ctstch -outDir clock_report -fixedInstBeforeCTS
Puts "Finished running CTS"

Puts "Timing the design after CTS"
timeDesign -postCTS -prefix postCTS_setup
timeDesign -postCTS -prefix postCTS_hold -hold

Puts "Setting Optimization Mode Option for DRV fixing"
setOptMode -fixFanoutLoad true
setOptMode -addInstancePrefix postCTSdrv

Puts "Optimizing for DRV"
optDesign -postCTS -drv

Puts "Timing design after DRV fixes"
timeDesign -postCTS -prefix postCTS_setup_DRVfix
timeDesign -postCTS -prefix postCTS_hold_DRVfix -hold

Puts "Setting Optimization Mode Option for Setup fixing"
setOptMode -addInstancePrefix postCTSsetup

Puts "Optimizing for Setup"
optDesign -postCTS

Puts "Timing design after Setup fixes"
timeDesign -postCTS -prefix postCTS_setup_Setupfix
timeDesign -postCTS -prefix postCTS_hold_Setupfix -hold

setOptMode -addInstancePrefix postCTShold
```

optDesign -postCTS -hold
Puts "Timing design after Hold fixes"
timeDesign -postCTS -prefix postCTS_setup_Holdfix
timeDesign -postCTS -prefix postCTS_hold_Holdfix -hold

Puts "Routing the Design"
setNanoRouteMode -quiet -timingEngine
setNanoRouteMode -quiet -routeWithSiPostRouteFix 0
setNanoRouteMode -quiet -routeTopRoutingLayer default
setNanoRouteMode -quiet -routeBottomRoutingLayer default
setNanoRouteMode -quiet -drouteEndIteration default
setNanoRouteMode -quiet -routeWithTimingDriven false
setNanoRouteMode -quiet -routeWithSiDriven false
routeDesign -globalDetail

Puts "Timing the design after Route"
timeDesign -postRoute -prefix postRoute_setup
timeDesign -postRoute -prefix postRoute_hold -hold

Exportar hacia Virtuoso
#set nameGDS "bwco_frame_BiCMOS_p2019.gds"; #Editar nombre de ser necesario

#streamOut ../GDS/$nameGDS -mapFile /media/Ext/libs/IBM_PDK/bicmos8hp/v.20171220/lef/bicm
-libName DesignLib -merge /media/Ext/libs/8HP_IP_CELL_AND_IO_Libs/BiCMOS8HP_Digital_Kit/il
/opt/libs/IBM_PDK/bicmos8hp/v.20160727/gds2/CMOS8HP_BASE_WB_IO_7LM.gds
-outputMacros -units 1000 -mode ALL

8.28.6 Filter basic

Script for RTL->Gate-Level Flow (generated from RC v12.10-s012_1)
VLSI Design Course
ITESO University
Davila Velade, Rene Saul; Ramos Contreras, Ricardo

puts "Hostname : [info hostname]"

##
Definitions for serch path attributes
##
set_attribute lib_search_path /media/Ext/libs/IBM_PDK/bicmos8hp/v.20171220/synopsys/
/
##
Setting the Target Technology Library (Library setup)
##

```
set_attribute library typ_v150_t025/PnomV1p50T025_STD_CELL_8HP_12T.lib

# Variable that points to the technology's .lef file
set my_lef_library /media/Ext/libs/IBM_PDK/bicmos8hp/v.20171220/lef/BICMOS8HP_SC_1P2


# set my_lef_library /media/Ext/libs/IBM_PDK/bicmos8hp/v.20171220/lef/bicmos8hp_5AM_21
/media/Ext/libs/IBM_PDK/bicmos8hp/v.20171220/lef/BICMOS8HP_SC_1P2V_12T_RVT.lef

# Next attribute controls the amount of information RTL produce
# when executing commands. The higher the value, the more verbose the output
(0-9)
set_attribute information_level 9 /

##################################################
## Load Design (HDL files)
##################################################

read_hdl ../HDL/AUDIO_DAC.v ../HDL/I2C_Controller.v ../HDL/serialRom_bandpass.v
../HDL/serialRom_lowpass.v ../HDL/simple_dual_port_ram_single_clock.v ../HDL/counter.v
../HDL/Filtering.sv ../HDL/Fixed_Point_MAC.v ../HDL/I2C_Module.v ../HDL/LRDecoder.v
../HDL/newCoefficient.v ../HDL/param_mux.v ../HDL/register.v ../HDL/registerChain.v
../HDL/Reset_Delay.v ../HDL/serial_delay.v ../HDL/serialFilter.v ../HDL/serialMAC.v
../HDL/shift_Register.v ../HDL/single_bit_param_demux.v

##################################################
## Performing Elaboration
##################################################

elaborate serialFilter

##################################################
## Constraints Setup
##################################################

read_sdc ../Filter_basic.sdc

puts "######################"
puts "Timing -lint Report"
puts "######################"

report timing -lint

##################################################
## Performing Synthesis
## Synthesizing to generic
##################################################
```

```
synthesize -to_generic -eff low

write_hdl > Filter_generic.v

report gates > Filter_gates_generic.rpt
report area > Filter_area_generic.rpt
report qor > Filter_qor_generic.rpt
report datapath > Filter_datapath_generic.rpt
report messages > Filter_messages_generic.rpt
report power > Filter_power_generic.rpt

################################################################
## Performing Synthesis
## Synthesizing to gates
################################################################
synthesize -to_mapped -eff low -no_incr

write_hdl > Filters_m_noincr.v

report gates > Filter_gates_m_noincr.rpt
report area > Filter_area_m_noincr.rpt
report qor > Filter_qor_m_noincr.rpt
report datapath > Filter_datapath_m_noincr.rpt
report messages > Filter_messages_m_noincr.rpt
report power > Filter_power_m_noincr.rpt
################################################################
## Performing Synthesis
## Incremental Synthesis
################################################################

synthesize -to_mapped -eff low -incr

write_hdl > Filter_m.v
report gates > Filter_gates_m.rpt
report area > Filter_area_m.rpt
report timing > Filter_timing_m.rpt
report qor > Filter_qor_m.rpt
report datapath > Filter_datapath_m.rpt
report messages > Filter_messages_m.rpt
report power > Filter_power_m.rpt
```

8.28.7 Filter simple

```
#### Template Script for RTL->Gate-Level Flow (generated from RC RC14.20
- v14.20-p005_1)
## ITESO University
## Ricardo Ramos Contreras
```

This RC Template was generated using the write_template -simple -outfile ¡Filter_simple.tcl> command

if [file exists /proc/cpuinfo]
sh grep "model name" /proc/cpuinfo
sh grep "cpu MHz" /proc/cpuinfo

puts "Hostname : [info hostname]"

##
Preset global variables and attributes
##

set DESIGN serialFilter
set SYN_EFF medium
set MAP_EFF high
set DATE [clock format [clock seconds] -format "set _OUTPUTS_PATH outputs_$DATE
set _REPORTS_PATH reports_$DATE
set _LOG_PATH logs_$DATE

Variable to specify the technology .lib file name
set timing_library typ_v150_t025/PnomV1p50T025_STD_CELL_8HP_12T.lib

#set my_lef_library /media/Ext/libs/IBM_PDK/bicmos8hp/v.20171220/lef/BICMOS8HP_SC_1P

set my_lef_library /media/Ext/libs/IBM_PDK/bicmos8hp/v.20171220/lef/bicmos8hp_5AM_21_te
/media/Ext/libs/IBM_PDK/bicmos8hp/v.20171220/lef/BICMOS8HP_SC_1P2V_12T_RVT.lef

set_attribute lib_search_path /media/Ext/libs/IBM_PDK/bicmos8hp/v.20171220/synopsys/
/

set_attribute script_search_path .. /

set_attribute hdl_search_path ../HDL /

##Default undriven/unconnected setting is 'none'.
##set_attribute hdl_unconnected_input_port_value 0 — 1 — x — none /
##set_attribute hdl_undriven_output_port_value 0 — 1 — x — none /
##set_attribute hdl_undriven_signal_value 0 — 1 — x — none /

##set_attribute wireload_mode ¡value> /

set_attribute information_level 9 /

```
##############################################################
## Library setup
##############################################################


set_attribute library $timing_library

## PLE
set_attribute lef_library $my_lef_library /

# If you want to ungroup change none to both
set_attribute auto_ungroup none /

##############################################################
## Load Design
##############################################################


read_hdl Filter_m.v

elaborate $DESIGN
puts "Runtime & Memory after 'read_hdl'"
timestat Elaboration

check_design -unresolved

##############################################################
## Constraints Setup
##############################################################

read_sdc ../Filter_basic.sdc

puts "The number of exceptions is [llength [find /designs/$DESIGN -exception *]]"

if ![file exists $_OUTPUTS_PATH]
file mkdir $_OUTPUTS_PATH
puts "Creating directory $_OUTPUTS_PATH"


if ![file exists $_REPORTS_PATH]
file mkdir $_REPORTS_PATH
puts "Creating directory $_REPORTS_PATH"



if ![file exists $_LOG_PATH]
file mkdir $_LOG_PATH
puts "Creating directory $_LOG_PATH"
```

To turn off sequential merging on the design
uncomment & use the following attributes.
##set_attribute optimize_merge_flops false /
##set_attribute optimize_merge_latches false /
For a particular instance use attribute 'optimize_merge_seqs' to turn off sequential merging.

```
################################################################
## Synthesizing to generic
################################################################
synthesize -to_generic -eff $SYN_EFF
puts "Runtime & Memory after 'synthesize -to_generic'"
timestat GENERIC
report datapath > $_REPORTS_PATH/$DESIGN_datapath_generic.rpt
generate_reports -outdir $_REPORTS_PATH -tag generic
summary_table -outdir $_REPORTS_PATH

################################################################
## Synthesizing to gates
################################################################

synthesize -to_mapped -eff $MAP_EFF -no_incr
puts "Runtime & Memory after 'synthesize -to_map -no_incr'"
timestat MAPPED
report datapath > $_REPORTS_PATH/$DESIGN_datapath_map.rpt

## ungroup -threshold ¡value>


################################################################
## Incremental Synthesis
################################################################

## Uncomment to remove assigns & insert tiehilo cells during Incremental synthesis
##set_attribute remove_assigns true /
##set_remove_assign_options -buffer_or_inverter ¡libcell> -design ¡design—subdesign>
##set_attribute use_tiehilo_for_const ¡none—duplicate—unique> /
synthesize -to_mapped -eff $MAP_EFF -incr
generate_reports -outdir $_REPORTS_PATH -tag incremental
summary_table -outdir $_REPORTS_PATH

puts "Runtime & Memory after incremental synthesis"
timestat INCREMENTAL
```

```
##################################################################
## Spatial mode optimization
##################################################################

## Uncomment to enable spatial mode optimization
##synthesize -to_mapped -spatial

write_design -basename $_OUTPUTS_PATH/$DESIGN_m
## write_hdl > $_OUTPUTS_PATH/$DESIGN_m.v
## write_script > $_OUTPUTS_PATH/$DESIGN_m.script
write_sdc > $_OUTPUTS_PATH/$DESIGN_m.sdc

report qor

puts "Final Runtime & Memory."
timestat FINAL
puts "============================"
puts "Synthesis Finished ....."
puts "============================"

file copy [get_attr stdout_log /] $_LOG_PATH/.

##quit
```

8.28.8 Filter simple BiCMOS

```
#### Template Script for RTL->Gate-Level Flow (generated from RC RC14.20
- v14.20-p005_1)
## ITESO University
## Ricardo Ramos Contreras
## This RC Template was generated using the write_template -simple -outfile ¡Fil-
ter_simple.tcl> command

if [file exists /proc/cpuinfo]
sh grep "model name" /proc/cpuinfo
sh grep "cpu MHz" /proc/cpuinfo


puts "Hostname : [info hostname]"

##################################################################
## Preset global variables and attributes
##################################################################
```

```
set DESIGN serialFilter
set SYN_EFF medium
###set MAP_EFF medium
set MAP_EFF high
set DATE [clock format [clock seconds] -format "%b%d-%T"]
set _OUTPUTS_PATH outputs_$DATE
set _REPORTS_PATH reports_$DATE
set _LOG_PATH logs_$DATE

# Variable to specify the technology .lib file name
set timing_library typ_v150_t025/PnomV1p50T025_STD_CELL_8HP_12T.lib

# Variable to specify the LEF library
set my_lef_library /media/Ext/libs/IBM_PDK/bicmos8hp/v.20171220/lef/bicmos8hp_5AM_21_te
/media/Ext/libs/IBM_PDK/bicmos8hp/v.20171220/lef/BICMOS8HP_SC_1P2V_12T_RVT.lef



##set ET_WORKDIR ¡ET work directory>
set_attribute lib_search_path /media/Ext/libs/IBM_PDK/bicmos8hp/v.20171220/synopsys/
/

set_attribute script_search_path .. /
set_attribute hdl_search_path ../HDL /
##Uncomment and specify machine names to enable super-threading.
##set_attribute super_thread_servers ¡machine names> /

##Default undriven/unconnected setting is 'none'.
##set_attribute hdl_unconnected_input_port_value 0 — 1 — x — none /
##set_attribute hdl_undriven_output_port_value 0 — 1 — x — none /
##set_attribute hdl_undriven_signal_value 0 — 1 — x — none /


##set_attribute wireload_mode ¡value> /
set_attribute information_level 9 /

################################################################
## Library setup
################################################################


set_attribute library $timing_library

## PLE
set_attribute lef_library $my_lef_library /

## Provide either cap_table_file or the qrc_tech_file
```

```
##set_attribute cap_table_file ¡file> /
#set_attribute qrc_tech_file ¡file> /

set_attribute auto_ungroup none /
#set_attribute auto_ungroup both /

##########################################################
## Load Design
##########################################################

read_hdl mux_2to1_1_bits.v mux_3to1_1_bits.v shift_Register.v serial_delay.v division.v
newCoefficient.v counter.v serialRom_lowpass2.v serialRom_lowpass.v registerChain.v
LRDecoder.v Fixed_Point_MAC.v serialMAC.v OneShot.v register_w_initial_conditions.v
registerFile.v i2c_slave.v adaptiveFilter_state_machine.v adaptiveFilter_i2c_controller.v
serialRom_bandpass2.v mux_8to1_16_bits.v serialRom_bandpass.v register.v param_mux.v
single_bit_param_demux.v  mux_16to1_16_bits.v  simple_dual_port_ram_single_clock.v
serialFilter.v

#

elaborate $DESIGN
puts "Runtime & Memory after 'read_hdl'"
timestat Elaboration

check_design -unresolved

##########################################################
## Constraints Setup
##########################################################

read_sdc ../Filter_opt1.sdc
puts "The number of exceptions is [llength [find /designs/$DESIGN -exception *]]"


if ![file exists $_OUTPUTS_PATH]
file mkdir $_OUTPUTS_PATH
puts "Creating directory $_OUTPUTS_PATH"


if ![file exists $_REPORTS_PATH]
file mkdir $_REPORTS_PATH
puts "Creating directory $_REPORTS_PATH"


if ![file exists $_OUTPUTS_PATH]
file mkdir $_OUTPUTS_PATH
puts "Creating directory $_OUTPUTS_PATH"
```

```
if ![file exists $_LOG_PATH]
file mkdir $_LOG_PATH
puts "Creating directory $_LOG_PATH"



#### To turn off sequential merging on the design
#### uncomment & use the following attributes.
##set_attribute optimize_merge_flops false /
##set_attribute optimize_merge_latches false /
#### For a particular instance use attribute 'optimize_merge_seqs' to turn off sequential merging.



##################################################
## Synthesizing to generic
##################################################


synthesize -to_generic -eff $SYN_EFF
puts "Runtime & Memory after 'synthesize -to_generic'"
timestat GENERIC
report datapath > $_REPORTS_PATH/$DESIGN_datapath_generic.rpt
generate_reports -outdir $_REPORTS_PATH -tag generic
summary_table -outdir $_REPORTS_PATH



##################################################
## Synthesizing to gates
##################################################

synthesize -to_mapped -eff $MAP_EFF -no_incr
puts "Runtime & Memory after 'synthesize -to_map -no_incr'"
timestat MAPPED
report datapath > $_REPORTS_PATH/$DESIGN_datapath_map.rpt



##Intermediate netlist for LEC verification..
write_hdl -lec > $_OUTPUTS_PATH/$DESIGN_intermediate.v
write_do_lec -revised_design $_OUTPUTS_PATH/$DESIGN_intermediate.v -logfile
$_LOG_PATH/rtl2intermediate.lec.log > $_OUTPUTS_PATH/rtl2intermediate.lec.do

## ungroup -threshold ¡value>
```

```
##################################################
## Incremental Synthesis
##################################################

## Uncomment to remove assigns & insert tiehilo cells during Incremental syn-
thesis
##set_attribute remove_assigns true /
##set_remove_assign_options -buffer_or_inverter ¡libcell> -design ¡design—subdesign>
##set_attribute use_tiehilo_for_const ¡none—duplicate—unique> /
synthesize -to_mapped -eff $MAP_EFF -incr
generate_reports -outdir $_REPORTS_PATH -tag incremental
summary_table -outdir $_REPORTS_PATH

puts "Runtime & Memory after incremental synthesis"
timestat INCREMENTAL



##################################################
## Spatial mode optimization
##################################################

## Uncomment to enable spatial mode optimization
##synthesize -to_mapped -spatial


write_design -basename $_OUTPUTS_PATH/$DESIGN_m
## write_hdl > $_OUTPUTS_PATH/$DESIGN_m.v
## write_script > $_OUTPUTS_PATH/$DESIGN_m.script
write_sdc > $_OUTPUTS_PATH/$DESIGN_m.sdc


#####################################
### write_do_lec
#####################################


write_do_lec -golden_design $_OUTPUTS_PATH/$DESIGN_intermediate.v -revised_design
$_OUTPUTS_PATH/$DESIGN_m.v -logfile $_LOG_PATH/intermediate2final.lec.log
> $_OUTPUTS_PATH/intermediate2final.lec.do
##Uncomment if the RTL is to be compared with the final netlist..
##write_do_lec -revised_design $_OUTPUTS_PATH/$DESIGN_m.v -logfile $_LOG_PATH/rtl2final.lec.l
> $_OUTPUTS_PATH/rtl2final.lec.do
report qor

puts "Final Runtime & Memory."
timestat FINAL
```

```
puts "============================="
puts "Synthesis Finished ....."
puts "============================="

file copy [get_attr stdout_log /] $_LOG_PATH/

report design_rules
report clocks

##quit
```