Universidade Federal de São Carlos– UFSCar
Centro de Ciências Exatas e de Tecnologia– CCET
Departamento de Computação– DC
Programa de Pós-Graduação em Ciência da Computação– PPGCC

Hugo Almeida Sampaio

Bit-True simulation for word length optimization of digital signal processing blocks

São Carlos
2024

Hugo Almeida Sampaio

Bit-True simulation for word length optimization of digital signal processing blocks

Dissertação apresentada ao Programa de Pós-Graduação em Ciência da Computação do Centro de Ciências Exatas e de Tecnologia da Universidade Federal de São Carlos, como parte dos requisitos para a obtenção do título de Mestre em Ciência da Computação.

Área de concentração: Sistemas de Computação

Orientador: Paulo Matias

São Carlos
2024

Este trabalho é dedicado à Nike e Urânia, deusa da vitória e musa da ciência.

Acknowledgements

There are many to whom I'd like to acknowledge by its efforts to helping me directly or indirectly to accomplish this work. None of this would have been possible without all support. First and foremost, my advisor Paulo Matias which presented me the theme and provided guidance and support through each step, supporting and point out important aspects. The partnership throughout these years is becoming this interesting fruit, with more possibilities to come. Second, my philosophy family from New Acropolis, specially Gabriel Nocera, that helped me becoming a better man and regain Will and peace of mind to accomplish this journey. Third family and close friends that supported me during all this work, taking me out of the research for a while so I could refresh, and regain strength and focus to finish.

aspera ad Astra

Abstract

During the development of DSP systems to be implemented in ASICs or FPGAs, the conversion of floating-point signals and operands to fixed-point is an important and time-consuming task, as fixed-point has faster performance and lower power consumption. However, the floating-point to fixed-point conversion problem is NP-hard and has no ideal solution. This work explores a hardware simulation technique, also known as Bit-true, based on masking the fractional bits of each operator and signal of interest directly in its FPGA implementation. This technique had only been proposed in the literature but never implemented before this work. It allows flexibility in defining optimization criteria and avoids the rework required by other techniques of implementing the system first in software, and only then transforming it into an equivalent hardware architecture. The DSP system chosen as a case study is a wireless BPSK receiver, designed for IoT applications. The receiver blocks were optimized in isolation using the technique, resulting in reduced area and power consumption, and increased maximum operating frequency. The artifacts produced in this work are expected to aid future research in IoT, reducing the time and effort to obtain a refined system systhesis.

Keywords: Bit-true simulation, Fixed-Point refinement, Word-Length Optimization, FPGA, DSP, IoT.

Resumo

Durante o desenvolvimento de sistemas DSP a serem implementados em ASIC ou FPGA, a conversão de sinais e operandos de ponto flutuante para ponto fixo é uma tarefa importante e demorada, pois o ponto fixo tem desempenho mais rápido e menor consumo de energia, mas o problema de conversão ponto flutuante para ponto fixo é NP-difícil e não tem solução ideal. Este trabalho explora uma técnica de simulação de hardware, também conhecida como Bit-true, baseada no mascaramento de bits de partes fracionárias de cada operador e sinal de interesse diretamente em sua implementação FPGA. Essa técnica havia sido apenas proposta na literatura, mas nunca implementada antes deste trabalho. Ela permite flexibilidade para definir critérios de otimização e evita o retrabalho exigido por outras técnicas de implementar o sistema primeiramente em software, para apenas depois transformá-lo em uma arquitetura de hardware equivalente. O sistema DSP escolhido como estudo de caso é um receptor BPSK sem fio, pensado para aplicações IoT. Os blocos do receptor foram otimizados isoladamente utilizando a técnica, resultando em redução de área e consumo de energia, e aumento da frequência máxima de operação. Espera-se que os artefatos produzidos neste trabalho auxiliem pesquisas futuras em IoT, reduzindo o tempo e esforço para chegar a uma sintese refinada do sistema.

Palavras-chave: Bit-true simulation, FPGA, DSP, IoT.

List of Figures

List of Tables

List of Acronyms

BER Bit Error Rate

CORDIC Coordinate Rotation Digital Computer

DFT Discrete Fourier Transform

DRE Dynamic Range Estimation

DSP Digital Signal Processing

FFT Fast Fourier Transform

FIR Finite Impulse Response

FWL Fractional Word Length

IDCT Inverse Discrete Cosine Transform

IIR Infinite Impulse Response

IWL Integer Word Length

LTI Linear Time-Invariant

PER Packet Error Rate

SNR Signal to Noise Ratio

SQNR Signal to Quantised Noise Ratio

Contents

Chapter 1

Introduction

As technology evolves, the need for higher performance and lower power consumption led a migration of mixed-signal systems to digital processing systems where complex analog blocks where digitized and just the simpler analog ones remains to condition the signal for digital processing. The digital counterparts of complex analog blocks are smaller, faster, more flexible and power efficient but require a methodology to handle the finite precision of math operands and operations, due to bit number restriction and adaptation on math operations (MENARD et al., 2019b).

A math operation (addition, multiplication, subtraction, etc.) on a digital system limits the number of bits on input and output, which leads to deviation comparing to the same operation done with infinite precision operators. On digital systems, higher the number of bits to represent operands, higher the power consumption, time and precision of calculation (MENARD et al., 2019b).

A good Digital Signal Processing (DSP) system design is capable of handling calculation deviations with no system performance degradation, e.g. a good low pass filter keeps its bandwidth and attenuation characteristics (MENARD et al., 2019b). Looking at Figure 1, we can see the steps for the conversion and their goals. As a starting point of DSP development, operands, math operators and algorithms are built on computational software, with floating point operands. Then the refinement starts over algorithms (e.g., changing a DFT to a FFT), then operators (e.g., approximate the inverse of square root – as in Fast InvSqrt from DOOM game (SHAW, 2017)), ending on converting floating point operands to fixed point. An efficient and generic technique for float to fixed conversion is an open problem, as this task currently represents between 25% and 50% of all DSP system development time (ALGARABEL, 2016).

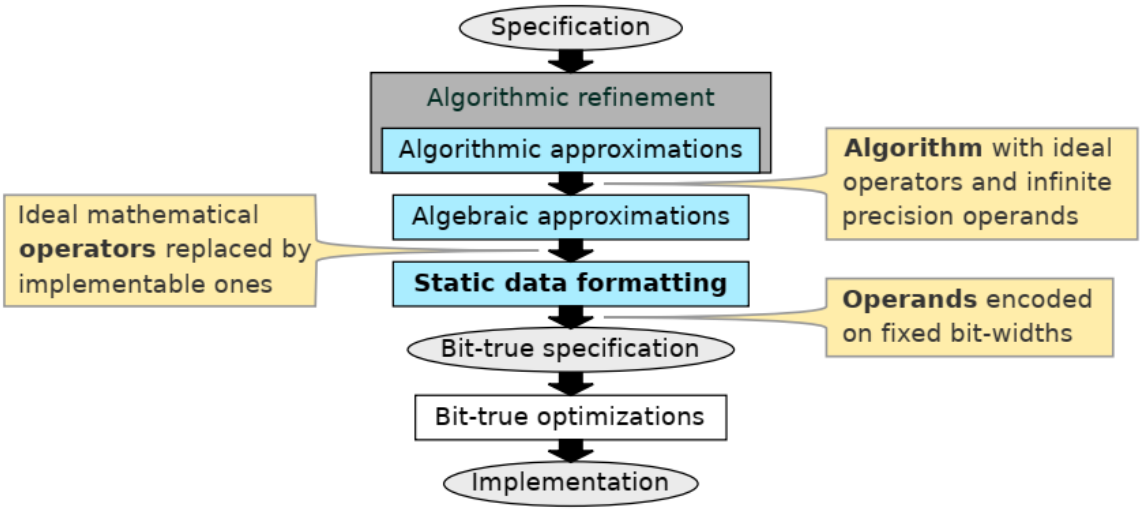

Figure 1 – Development phases. Source: Menard et al. (2019b)

Fixed Point is a binary representation where the separation of integer and fractional parts is virtual and selected according to the system needs, as shown in Figure 2. It can be signed or unsigned and have an arbitrary number of bits allocated to each part. A fixed point number has m bits to represent its integer value where its most significant bit (MSB) represents its sign, and n bits to represent its fractional value. Following this format, its resolution (also known as quantization step) is $q = 2^{-n}$ and the values that can be represented (its dynamic range) are between -2^{m-1} e $2^{m-1} - 2^{-n}$. Example: a signed number with m=5 and n=4 has quantization $q = 0.0625$ and can represent values from -16 to +15.9373. If it is unsigned, its resolution is the same $q = 0.0625$ but its dynamic range changes to 0 to $2^5 - 2^{-4}$, i.e. 0 to +31.9375. The "Q-Format" notation (Qm.n) can represent the number format: Q5.4 for the first example and uQ5.4 for the second one.

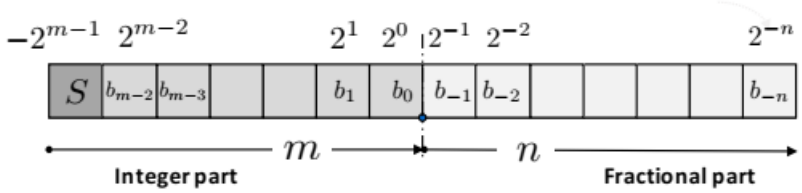

Figure 2 – Fixed point representation. Source: Menard et al. (2019b)

The float to fixed point conversion is typically split in 2 phases: integer word length (IWL) determination and fractional word length (FWL) determination – the whole process is shown on Figure 3.

During IWL phase, the main concern is to find the smallest m for each operand such that every operation do not result in overflow. IWL phase is divided on 2 steps:

- ❏ Dynamic Range Estimation (DRE), where the magnitude of each operand is estimated based on simulation (NEHMEH et al., 2014) or statistics (WU; ZHU; NAJM, 2006). The number of bits for integer parts are determined by $IWL = \lceil log_2 R \rceil$. where R is the biggest absolute value found for a given operator (HAN; EVANS, 2006).

- ❏ Scaling, to optimize virtual point placement and avoid "insignificant" bits on operations where m is too far apart.

The FWL phase looks for the smallest n value for each operand for which the system presents acceptable math operation errors. The process to accomplish this is a optimization algorithm with 2 constraints: accuracy and cost. Accuracy limits the maximum deviation tolerable for operations or the whole block – usually this restriction depends on the system, *e.g.* a image signal processing system might prefer SSIM and a DSP for RF might choose BER, PER or SQNR. The cost is usually obtained from implementation results, *e.g.* in a FPGA or ASIC we measure the resource count (LUTs, transistors, DSP blocks), silicon area, processing speed/latency, power consumption.

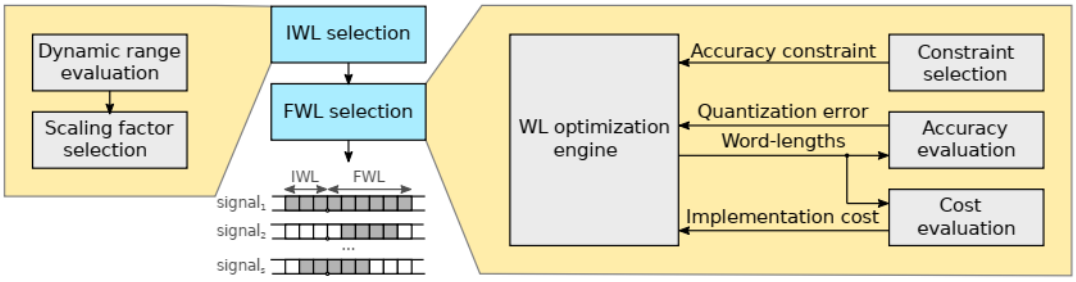

Figure 3 – Conversion process. Source: Menard et al. (2019b)

The techniques behind the algorithms for FWL optimization can be divided in 2 groups: analytical and simulation. Analytical techniques work by building equations based on statistics of the operands – each solution for the group of equations is a possible FWL solution which is validated against the constraints. The first versions such as Keding et al. (1998), Kum, Kang and Sung (1999), Kum and Sung (2000) were all designed for the C language, both because the designs were is C and because of performance. Many other works followed this idea, such as Constantinides, Cheung and Luk (2003), Osborne et al. (2007). Analytical techniques are faster but more conservative and usually only applicable to LTI systems or loop free systems (MENARD et al., 2019b).

Simulation techniques consume a dataset that corresponds to real inputs and iteratively change the word length of an operand, comparing then the result produced by the changed circuit to its "golden" version. The changes that best impact the system are kept for next iterations. The optimization stops when no more improvements are found within the given constraint.

Both techniques for IWL and FWL compare accuracy based on the system with floating point operands and also are designed to be computed on ordinary computer hardware (such as a PC or a server). No technique tries to integrate specialized hardware (FPGA or ASIC) in the loop.

1.1 Motivation and Objectives

The interest for studying word length optimization came from the goal of fostering the experimental research on RF transceiver implementation for IoT applications. Industrial scenarios are composed of machines that require constant communication with a supervisory system and they carry low bandwidth data such as digital inputs and output states, temperature, pressure, electrical current and voltage, pluviometric data, etc. Industrial applications also need to cover long distances (up to 10 km) and might also have no access to mobile phone network coverage. To face these challenges, efficient modulation schemes, mesh routing protocols and robust channel coding techniques are great enablers and the development of radios with such characteristics motivates this work. On the other hand, size, cost and low power restrictions drive the need of tools to help on optimization of the radio design and accelerate development. Based on this motivation, the examples and DSP blocks designed and optimized on this work are components of RF transceivers, such as filters, CORDIC algorithm, coarse and fine frequency estimators (and corrector) and time synchronizers.

The focus of this work is to follow a proposal due to Hormigo and Caffarena (2021), a method on which the FWL of each signal of a system is attached to a bitmask, limiting the effective number of bits prior to any math operation by a logical AND operation. The architecture from Hormigo and Caffarena (2021) has no implementation known by the author, therefore the contributions of this work are providing a reference implementation of Hormigo and Caffarena (2021)'s method and validating that the method can achieve the same results as a PC simulation. The indirect objective is to merge parts of "Static data formatting" with the "Bit-true optimization", shown in Figure 1, to accelerate the design time, as this process of reaching the optimal implementation takes up to 50% of the design time. Dynamic range for determination of IWL of each operand is simpler to obtain, requiring only one set of simulations on floating point and calculating the upper bound for each operand, so IWL will be set accordingly at FPGA level. As a reference for simulations and as a support tool for comparison, the system was first implemented using the float type (double precision) in Python3 and compared in terms of SQNR.

The design of the data frame was based on some ASIC available on the market, such as SPIRIT1, S2-LP and SX127x. A first information to compare is how these transceivers design a packet. Their datasheets (STMICROELECTRONICS, 2021; STMICROELECTRONICS, 2023; SEMTECH CORPORATION, 2020) show they have a packet mode

with a flexible frame, presented on Figures 4, 5 and 6.

1-32	1-4	0-16 bit	0-1	0-4	0-65535	0-3
Preamble	Sync	Length	Address	Control	Payload	CRC

Preamble (programmable field): the length of the preamble is programmable from 1 to 32 bytes by the PREAMBLE_LENGTH field of the PCKTCTRL2 register. Each preamble byte is a '10101010' binary sequence.

Sync (programmable field): the length of the synchronization field is programmable (from 1 to 4 bytes) through dedicated registers. The SYNC word is programmable through registers SYNC1, SYNC2, SYNC3, and SYNC4. If the programmed sync length is 1, then only SYNC word is transmitted; if the programmed sync length is 2 then only SYNC1 and SYNC2 words are transmitted and so on.

Figure 4 – Spirit1 packet frame. Source: STMicroelectronics (2021)

Table 51. BASIC packet format

Preamble	Sync	Length	Address	Payload	CRC	Postamble
0:2046 bits	0:32 bits	0:2 bytes	0:1 bytes	0:65535 bytes	0:4 bytes	0:510 bits

- **Preamble**: each preamble is a pair of '01' or '10' from 0 pair to 1023 pairs, programmed by the register PREAMBLE_LENGTH. The binary sequences transmitted in the various modulation modes are summarized in the following table (leftmost bit is transmitted first).

Figure 5 – S2-LP packet frame. Source: STMicroelectronics (2023)

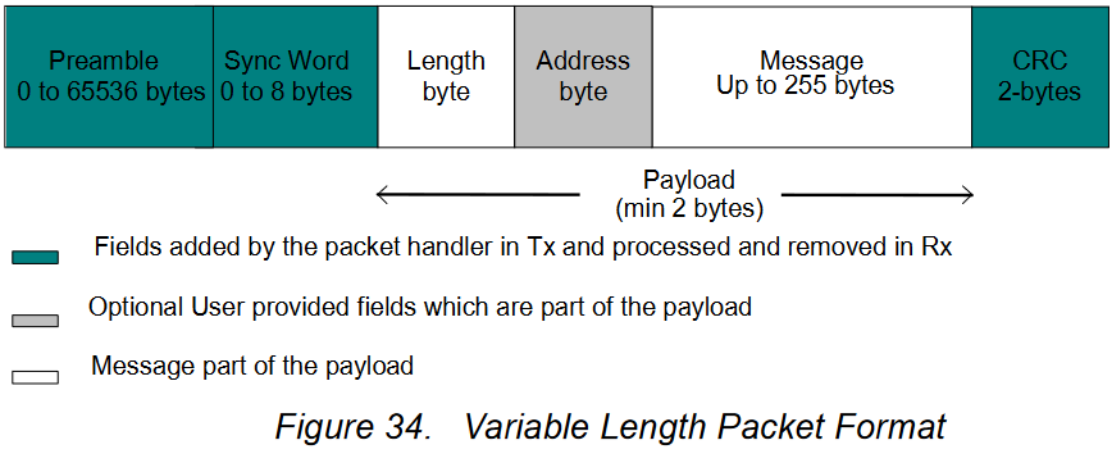

Figure 34. Variable Length Packet Format

Figure 6 – SX127x packet frame. Source: Semtech Corporation (2020)

All have a flexible preamble, composed of 1s and 0s, followed by a sync word to identify the start of frame. Spirit1 and S2-LP also have a quality indicator that evaluates

the sequence of 1s and 0s by counting the shifts to drop the reception of a packet if a certain number is not met. The sync word also has to perfectly match to allow the MAC to process the stream.

As the focus of this work is to implement a BPSK reception chain (detailed further on), the transmitted packet consists of 2 bytes for the preamble and 10 bytes for sync and data, and data is generated randomly for testing purposes. The processing chain will work on the preamble sequence to estimate the channel impairments and fix the whole packet.

Chapter 2

Literature review

Below a few articles and thesis are presented which guided understandings and decisions taken on this work. To date, the problem of float to fixed point conversion has been studied for at least 25 years.

Menard et al. (2019b) present a review of the float to fixed point conversion process, starting with the concept of fixed point, system design stages, fixed point processing architectures (from customized implementation for ASIC to strategies on processors with SIMD instructions), review of previous works to determine both IWL and FWL, from analytical ones (interval arithmetic, affine arithmetic, chaos polynomial among others) to simulations. Menard's work focuses on describing the techniques, their results and limitations.

Cherubin and Agosta (2020) build a catalog of tools for reduced precision computation, comparing different tools with similar approach. The purpose of this work is broad, comparing tools by the data type they use (not all results in fixed point), programming language accepted as input and generated as output, tool's main purpose (analysis/verification, optimization) and scope (annotated parts of input, the whole input, kernel blocks). All tools presented are software, mainly designed in C/C++ or extensions for LLVM compiler, where they extract operands and generate code optimized for some specific processor architecture.

Cantin, Savaria and Lavoie (2002) propose a comparison of 9 different word length optimization techniques, all based on simulations. They choose 12 DSP blocks (CORDIC, FIR, IIR, IDCT, etc.) and 2 metrics: the number of simulation steps required to reach a result and the distance from found result to optimal. The optimization techniques are:

- ❏ Min + b: 2 steps. First the minimum value for each operand is found keeping all

other unaltered (this is called MWL). Then all operands are configured to what MWL found (which should be below the accuracy constraint) and a round starts where 1 bit increments are applied to each operand, the bit which pushes accuracy closer to constraint is kept and new rounds are done until achieving the constraint. If $b = 1$, this algorithm is called sequential search, but $b = 1$ tends to get stuck in local minima and $b > 1$ is an improvement to avoid these local minima;

- Max - 1: All operands start with maximum number of bits (32 bits on Cantin, Savaria and Lavoie (2002) work). The round subtracts 1 bit at a time and the bit that shows the least improvement to constraint metric is kept and a new round starts. It stops when no more bits can be subtracted keeping the system above accuracy. This algorithm also follows Min + b tendency to reach local minima;

- Evolutive: Also has 2 steps. It start with all operands as floating point, each operand is configured with a minimum length and incremented until reaching constraint. After finding the value to all operands, they are configured with $b + 1$ and the "Max - 1" algorithm above is applied;

- Hybrid: "Max - 1" followed by "Min + b";

- Heuristic: It starts with all operands configured to MWL and increments 1 bit on all signals until reaching the constraints, then "Max - 1" is applied.

- Simulated Annealing: All operands start with a value such that the system is compliant to the constraint, then a operand increases p bits so all other can decrease r bits and reduce the cost.

- Preplanned: A 3 step technique. First the sensibility of the system $f(WL)$ is computed, where $E = f(WL+1) - f(WL)$ for each operand. Then a decreasement sensibility list is built and finally all operands are set to MWL and incremented based on sensibility list until reaching the constraints.

The authors conclude the work showing that for some systems there are optimal values better than MWL and there isn't an efficient general technique that works for every DSP block.

Rosa et al. (2016) design an evolutionary algorithm based on the compact genetic algorithm (cGA). The first stages compute the IWL and format the system for simulation. Them cGA is used to find sub-optimal values for each FWL operand. The article also presents a theoretical analysis proving it is sub-optimal. There is also a case study applying the technique to a DSP algorithm for robotics.

All the previously discussed works go by the same *modus operandi* of simulating or analysing the DSP design as a single block, doing all the work on a computer. Some changes to this *modus operandi* came on the following works.

Ha, Yuki and Sentieys (2020) optimize a Image Signal Processor (ISP), a large system composed of 8 blocks in a pipeline. The attribute chosen as a constraint is called Structural Similarity (SSIM) and the authors declare existing analytical structures are based on SNR/SQNR as constraint metric and can not be changed to quality metric such as SSIM. The approach proposed is to reduce the complexity of the system by optimizing each block individually based on a metric they called noise budge, the loss of quality for the whole system that can be applied to each block individually to obtain the same global result.

Hormigo and Caffarena (2021) describe a architecture to compute Bit-True simulations using a FPGA and a PC, where input signals, the comparison to reference output and a control unit blocks are allocated on FPGA, the PC decides which operand to optimize and communicates to the control unit, which then runs simulations and returns the results back to the PC. They also propose a limiting precision mask to clear the LSB of each FWL operand so that the FPGA system do not need new synthesis for each round of simulation. The process of synthesis is time consuming and by avoiding it, simulations on FPGA become possible. This work is purely a proposal, the authors do no show any implementation nor practical results. They also suggest that such architecture can be used to make searches such as proposed by Cantin, Savaria and Lavoie (2002).

Finally, citing related work on the subject of transceiver design, Greisen, Haene and Burg (2010) document the process of design flow, verification and characterization for an industrial-grade for IEEE802.11n compliant MIMO OFDM baseband transceiver. They describe bit-true simulations both on software and FPGA for different phases and purposes for verification and characterization of the transceiver and include BER as part of the metrics for testing and validating the fixed point stage of the design. All measurements are made by Monte Carlo simulations. However, the authors employ an approach of manually converting MATLAB code to RTL during a step of their flow, and claim there exists no single language "equally well suited for all design representations."

Ng (2011), Fleming et al. (2011) design a modular OFDM baseband transceiver on FPGA capable of reaching reliable communications up to 24Mbps, following the 802.11g standard as a baseline. The result put as the central innovation of Ng's Ph.D. thesis is showing that transceiver implementations, although heavily dependent on DSP blocks, can perform better when designed in a high-level HDL which preserves control over hardware micro-architecture (such as Bluespec SystemVerilog) when compared to C-based HLS tools. This showcases a different point of view than Greisen et al., in that Bluespec or similar languages could actually be well suited for design representation at every development stage. However, Ng et al. don't tackle the issue of fixed point optimization, leaving the designer as responsible for setting the bit width.

Chapter 3

Development

All code described in this chapter can be found at <https://github.com/hugoasampaio/hueblue-fw>.

3.1 Of what went wrong

The first idea to implement an operand whose FWL is limited by a bit-mask came from object oriented programming: an object that can be instantiated with any type but always has a bit-mask attached to it. The type itself might be a complex fixed point or a real fixed point and its usage might be as a constant (*e.g.*: an array of filter taps) or as registers (*e.g.*: auxiliary variables inside a convolution block). Following this approach, the result would be a simulation tool which a designer could use to create their DSP blocks with no modification and simulate the FWL values directly on hardware if desired. After finding the optimal values, the final system would be obtained simply by replacing them back on type declarations in the source code, a job which could be fully automated if each limiter had some sort of identifier linked to its order of appearance at source code.

An approach to apply that idea would be to analyse and modify the abstract syntax tree (AST), as a source code pre-processor. As every node on the tree is data, the approach would be to modify the nodes to add the bit-mask operation, but this comes with a challenge: adding the Control Unit to these bit-mask operations, as they are not static. Each bit-mask changes the number of bits masked during the simulation turns. Every Control Unit needs to be routed across module hierarchy all the way up to the top module, and this would require changing the module interfaces of an entire design. It's not trivial to create a tool able to make such global modifications at the AST level.

Another approach is to employ language features and implement the operand from inside the HDL. Bluespec offers ModuleCollect, which abstracts a second control interface for a module and allows the collected group of control interfaces to be opened and explored on a third module designed to handle these interfaces. Under ModuleCollect, each limited variable would be an instance of a module, named LimitedVar. It would have two interfaces: a control interface to set the number of FWL bits to be zeroed and a data interface to access the signal after limiting its FWL. The module itself just applies the bit-masking based on a number received on the control interface. The control interface of each instance would be collected by ModuleCollect and opened on a main Control Unit. The data interface would be applied on other modules to build the DSP and arithmetic blocks.

Although the ModuleCollect approach would work fine for registers holding a fixed point value to be limited by a bit-mask, it would not be ergonomic to use for intermediate values in a combinational circuit. As a pure functional language, Bluespec represents combinational circuits as pure functions, which cannot modify state elements (such as registers or memory) nor instantiate modules. This means it would not be possible to declare a LimitedVar inside a function. The LimitedVar would need to be instantiated inside a module, and the function accessing it would need to be declared inside that module. It would not be possible to share that function between different modules (*e.g.*, as a library). Moreover, one would need to change attribution syntax to use the "`<=`" operator or a method call instead of using the ordinary "`=`" operator, which would conceal the combinational nature of the variable. Last but not least, two different modules implementing the LimitedVar interface would exist – one for combinational variables and another one for registers.

Two other possible high-level HDLs which, in the same spirit of Bluespec, preserve designer control over hardware micro-architecture, are Chisel and SpinalHDL. Actually, SpinalHDL is a fork of Chisel, so both share a common ground: an "extension" to Scala programming language, which has some functional programming features (like Bluespec and its base, Haskell) and is built upon Java. Just like Bluespec, these languages compile to Verilog which can be fed to standard synthesis tools. As SpinalHDL has a better support for fixed point types, we chose it as another potential target for designing our limited fixed point type.

SpinalHDL has the AFix type, which encapsulates the fixed point idea with methods to create an object based on different specifications, define rounding and truncation, carry the arithmetical operations and provide access to the raw (binary) value behind it. So it gives the same level of access as Bluespec, plus the possibility of encapsulating AFix in another class which would manage the masking operation and the value of each mask. The diagram of Figure 7 shows the implementation of the raised cosine filter for the BPSK transmitter.

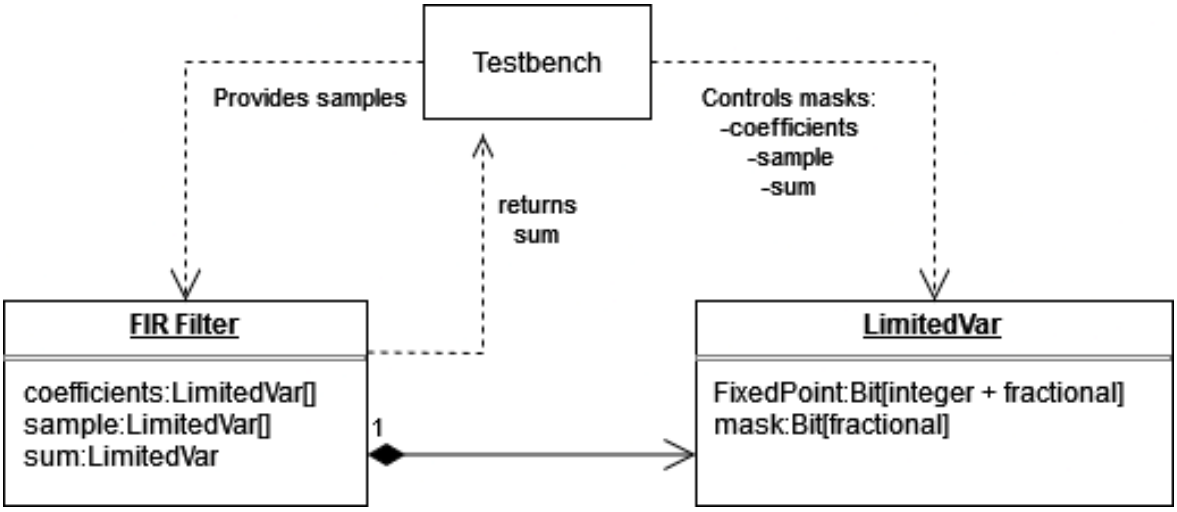

Figure 7 – BPSK transmitter blocks designed for SpinalHDL. Source: Author

But we were not able to devise a transparent way to allow a block to have an extra control interface for FWL on SpinalHDL nor on Chisel. The first issue found is the strict dependency on SpinalHDL and Chisel where a parent class can only access their direct descendants. The design tested had a test-bench as top class, a convolution block as child and the LimitedVar class (implementing Fixed-Point with AFix type on SpinalHDL) as grandchild. The top class could not access the instances of LimitedVar objects inside the FIR filter for controlling the mask values, even if the FIR Filter block had an interface to allow it. The only solution found was adding the FWL bit-masking operation to the convolution block, but this would break the idea of requiring minimal patching of the source code to carry out bit optimization.

In the end, we decided to return to Bluespec, because the experience gained during the tests described above showed it had a more comprehensible Fixed Point library. However, no further effort was exerted on improving coding ergonomics, focusing instead on implementing functioning transceiver blocks, such that this work could serve to assess the viability of the bit-masking idea. Thus, for the final implementation, the DSP blocks were designed with the FWL bit-masking operation explicitly applied at the source code. To control the mask values, however, we retained some ergonomics by employing CBus, a module from Bluespec standard library which employs ModuleCollect under the hood to automatically route control registers to the top module.

Since the partial LimitedVar implementations in SpinalHDL may have value for future work aiming to improve coding ergonomics, the source code is also available at the author's GitHub repository.

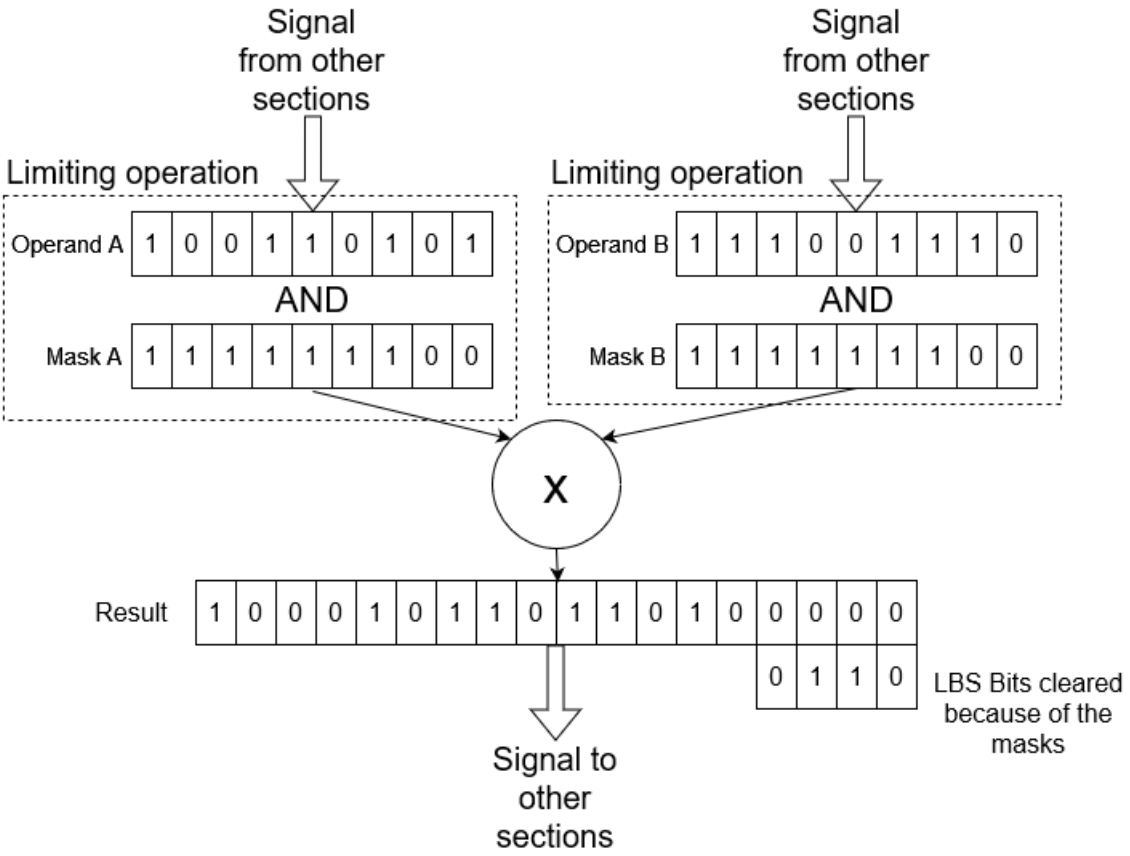

Figure 8 – Limiting Operation.

3.2 Architecture Overview

This sections describes some details about how the work was designed, the steps followed during its development, what exactly is the bitmasking operation proposed by Hormigo and Caffarena (2021), the code organization on author's GitHub and some other architectural details.

3.2.1 Bitmasking/limiting

Bitmask/Bitmasking or limit/limiter/limiting are words used interchangeably in this work and represent the same idea during a math operation, shown in Figure 8, on which we have 2 signals coming from some section on the DSP processing chain that need to be multiplied. The bitmasking/limiting operation is the step of passing each signal through a logical AND operation with a masking word before the math operation. It's important that the LSB of the mask is aligned with the LSB of the signal. In this work, limiting is done with the fractional word of each signal where the mask has the same size of FWL. The Figure 8 shows two operands of 9 bits each and its masks, A and B, each clearing 2 LSB from the signals. The operation requires 18 bits and its 4 LSBs are cleared because of masking.

3.2.2 Design steps and validations

The design of this work was done following the steps on Figure 1, but starting at the "Static data formatting" phase. The indirect objective is to merge parts of "Static data formatting" with the "Bit-true optimization" by confirming that fixed-point optimization can be done during "Bit-true optimization" step, directly on hardware. To do so, this work designs 3 implementations of the same algorithms: first a "gold standard" on Python, then its Bit-true bitmasked BSV version and finally hardware synthesized version. Each implementation is validated by its predecessor and validates its successor. As each step needs validation before proceeding to next one, these are the steps and their validations:

- ❏ Python designs with double type as "gold standard" for all other blocks. There is no formal validation but the results are verified graphically, observing the signal shapes compared to literature and to reference taken from the <https://pysdr.org> digital book;

- ❏ BSV designs with bitmasking, also tested on PC, are compared with Python standard through SQNR, graphical inspection of only the optimal result, as shown on graphs during the next sections;

- ❏ Hardware-in-the-Loop(HIL) simulation of the BSV design are compared to BSV designs with bitmasking on PC so we can validate that simulating bitmasking on PC or on hardware have the same results.

Looking through the simulations perspective, there are 2 steps done as PC simulations and a last validation on hardware. "Python" and "BSV limited" are all PC simulations, and "HIL" comes as last step to check whether "HIL" and "BSV limited" are equivalent and can be merged as only "HIL," fulfilling the indirect objective presented in the Objectives section.

3.2.3 Code organization and synthesis tool

The code at <https://github.com/hugoasampaio/hueblue-fw> contains all the Python and BSV code for this work, where each algorithm is a folder. Each folder is self-contained and has BSV code for simulation and synthesis, Makefile script and Python code for simulation. BSV allows creating a cycle accurate executable based on the code generated and compiled by Verilator. Also allows creating just the Verilog so it can be exported to a synthesis tool. The FPGA chosen for this work is a Xilinx's Spartan6 XC6SLX16 because the author could find a straighforward and well documented board available, but the Spartan6 family was discontinued by Xilinx so the synthesys tool for Spartan6 family is ISE 14.6, a discontinued tool.

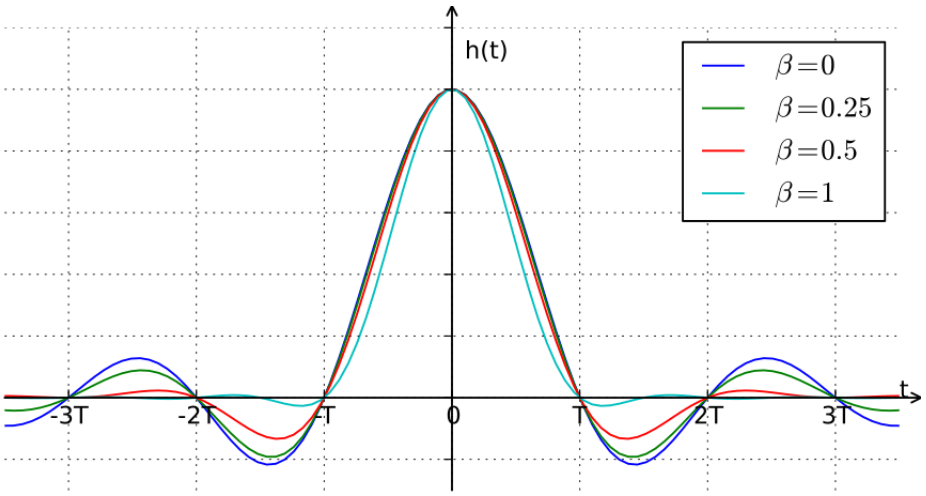

Figure 9 – Raised cosine Impulse Response. Source: <https://en.wikipedia.org/wiki/Raised-cosine_filter>

3.3 The BPSK processing chain

Before introducing the baseband reception chain implemented in this work, let's introduce the Figure 10 showing the whole chain for a generic wireless communication system, from the arrival of a frame to the transmitter until the delivery of a frame to the receiver. This work is focused on the first 3 yellow blocks of the chain: Coarse Frequency Synchronization, Time Synchronization and Fine Frequency Synchronization, but other blocks are also designed as Python simulations, since they are necessary to conduct the experiment and generate results for comparison. The Modulation and Pulse shaping are implemented in Python, composing the BPSK transmitter, and the Wireless Channel corresponds to the 3 channel impairments: AWGN, time and frequency shifts.

The modulation block carries out Binary Phase Shifting Keying, where each bit represents a phase value for the carrier. The Pulse Shaping is a technique that avoids Inter-Symbol Interference (ISI), by shaping the energy spectrum of a pulse in a manner to minimize energy on sampling periods. This can be seen on the zeros the function achieves at each period T on Figure 9 – this means the energy is concentrated at the instant the symbol is sampled (the 0 in time axis) and its energy is zero at the instants on which future and past signals are sampled (the T's in time axis). There are different types of filters which can present this property; a common one is the Raised Cosine, chosen for this work. The Matched Filter is a matching pair of the Pulse Shaping filter; since the Wireless Channel distorts the signal, the Matched Filter helps maximizing the SNR at the receptor by correlating the signal with the Pulse Shaping used by the transmitter. To keep the ISI canceling capability, the Raised Cosine filter is split into two Root Raised Cosine filters, so after convolving the signal twice (at transmitter and receptor) the signal keeps ISI and pulse shape at the receptor.

The system chosen as a problem to be optimized is a BPSK demodulation chain, composed of a Delay and Multiply algorithm for coarse frequency synchronization, Mueller & Miller timing synchronization algorithm and a Costas Loop fine frequency synchronization. Both transmitted and received signals are treated as baseband: all passband processing is out of the scope of this work.

First, the entire chain of transmission and reception was implemented in Python with floating point operands and graphics to follow the signal transformation. The full signal transformation chain is composed of:

- A sync signal composed of 16 bits, followed by 10 bytes of random bits representing the packet to transmit;
- Then the signal is interpolated and convoluted with a 42 tap raised cosine pulse to generate a train of pulses compliant to Nyquist's criteria. These 2 steps convey a transmitted packet. The diagram for these steps can be found on Figure 11 below, where there is a description of each block and a graphic with its result. Real samples are represented in blue, and imaginary samples are represented in orange. The baseband signal, ready for passband conversion and transmission, has no imaginary values;
- To simulate the channel, 3 impairments are applied – first AWGN, then fractional time delay and finally frequency offset, found on Figure 12. Channel impairments scramble the real samples and add values to imaginary samples;
- Receiver stage starts with delay and multiply frequency estimator and a secondary block to apply the frequency offset fix, whose diagram is shown on Figure 13. Here part of the energy comes back to real samples and part of its shape is recovered;
- 2nd stage is Mueller and Muller timing synchronization, which also applies decimation over the signal, as shown in Figure 14. Decimation chooses the best samples to represent each symbol and greatly reduces imaginary on energy samples;
- 3rd stage is a Costas loop fine frequency correction and its result is the original PAM transmitted signal, represented on Figure 15. Finally, a bit more energy from imaginary samples is removed, such that the signal is mostly real again.

This chain was based on Dr. Marc Lichtman's digital book <https://pysdr.org/content/sync.html>. The coarse frequency estimator from Dr. Marc Lichtman is based on FFT transform, which we decided to avoid in order to reduce FPGA resource usage, so a delay and multiply algorithm was chosen from Mengali (1997). That algorithm is commonly used as coarse estimator on data-aided systems, which is the proposal of this work. All mathematical background behind each block of this work is also detailed in Mengali (1997) book.

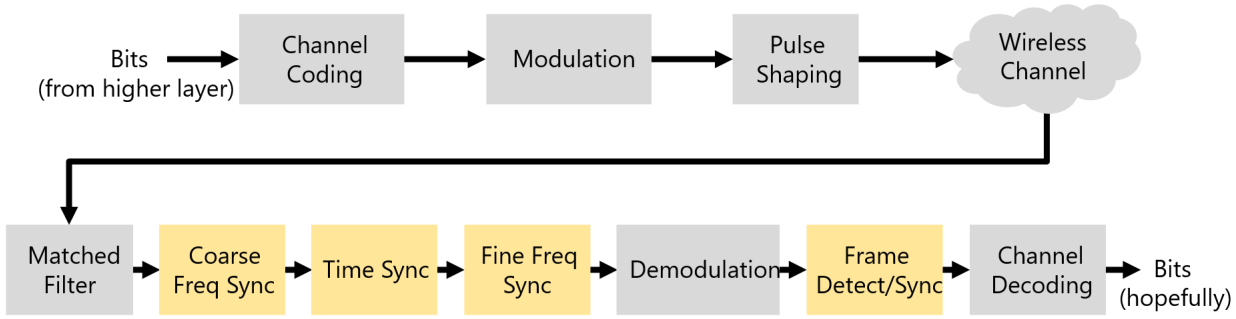

Figure 10 – Wireless communication chain. Source: <https://pysdr.org/content/sync.html>

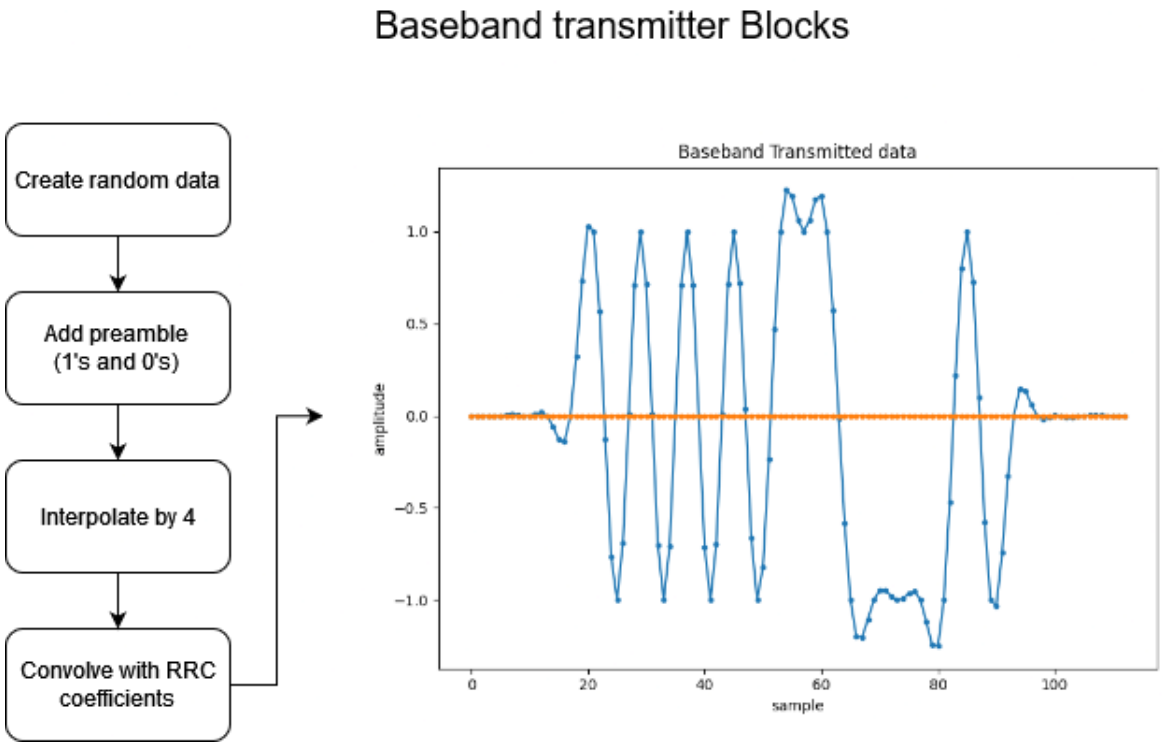

Figure 11 – BPSK transmitter blocks designed on Python. Source: Author

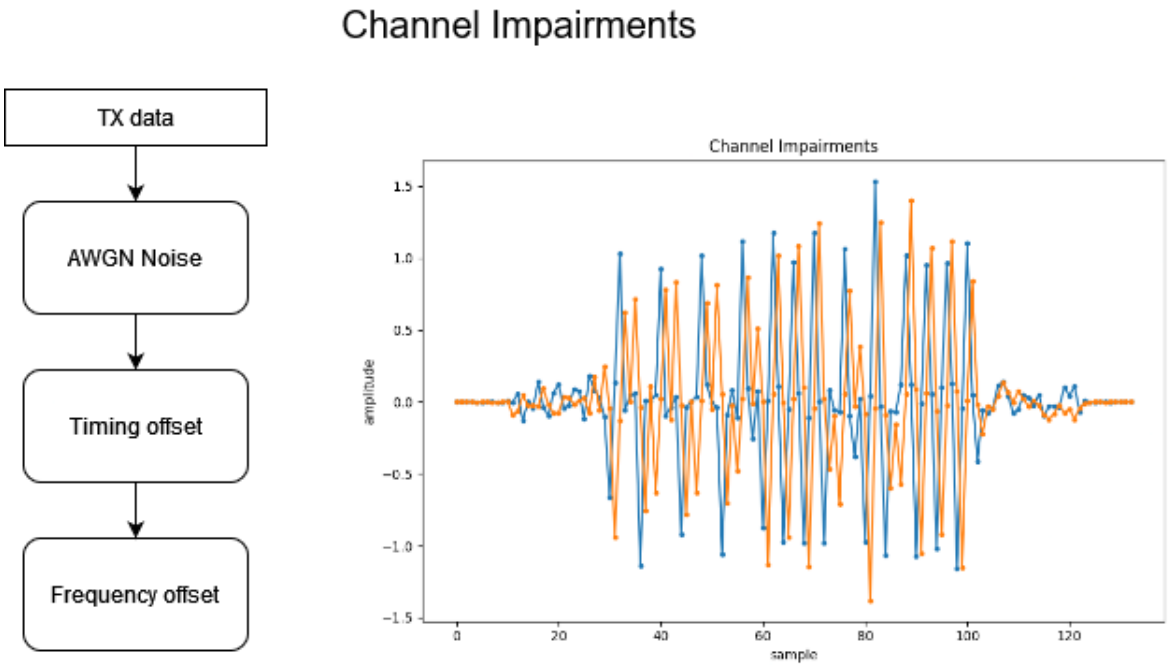

Figure 12 – Channel impairments designed on Python. Source: Author

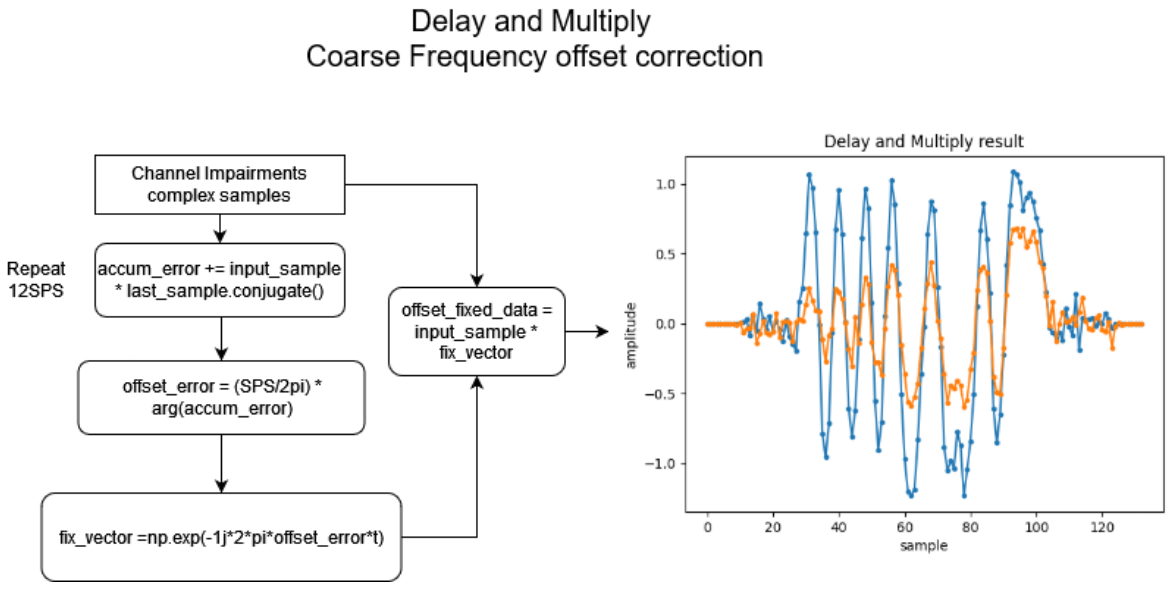

Figure 13 – Delay and Multiply algorithm designed on Python. Source: Author

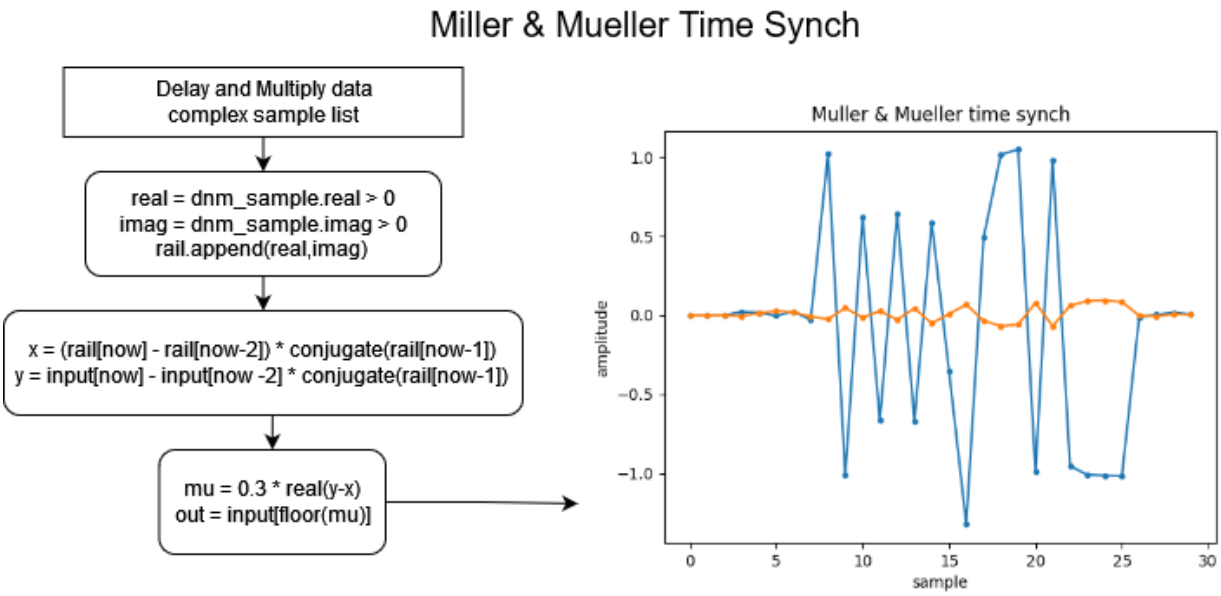

Figure 14 – Muller and Mueller designed on Python. Source: Author

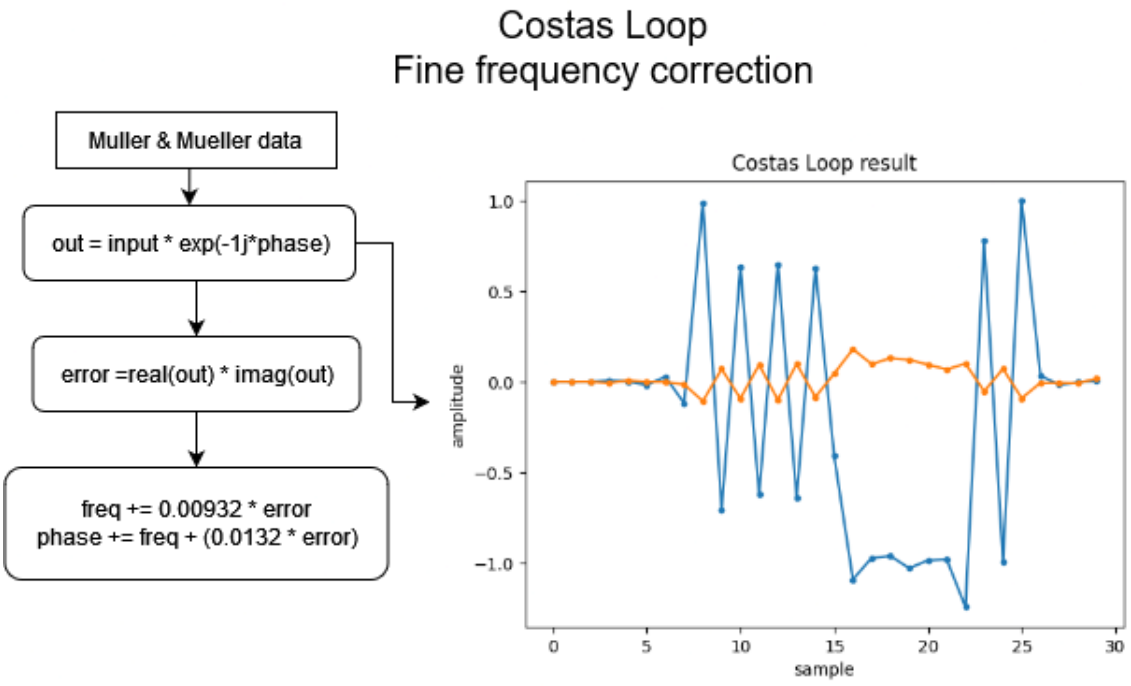

Figure 15 – Costas Loop designed on Python. Source: Author

3.4 Bluespec implementation

The second stage was to develop the same receiver chain using Bluespec System Verilog (BSV), where all signals and operands were defined as fixed point. BSV has support for Real and Complex fixed point numbers so basic math operations like add and multiply were already designed. BSV also accepts declaring static Real and Complex numbers as fixed point and it automatically converts during compile time so all constants and coefficients could be effortlessly copied from Python.

The first big difference came from arc tangent calculation required for delay and multiply coarse frequency estimation. Python has `atan2` function, which returns an accurate arc tangent. For the Bluespec implementation, we designed an atan mode CORDIC.

The second difference is on coarse frequency corrections. Python creates a complex vector with the carrier difference and multiplies it by the received vector. To create such vector, a CORDIC algorithm on rotate mode was designed based on the simplification of a table of $tan2^{-i}$ angles (which can perform rotations from -90 to +90 degree) and an extra loop operation of 45 degree rotation for input angles whose magnitude is above 90 degree. To create the fix vector, the CORDIC is looped starting with a complex $1 + j0$ and the error estimated – each output is a sample that corrects the received signal and the input for the next step on CORDIC loop.

Mueller and Muller timing synchronization was implemented with no changes when compared to the Python implementation. Costas loop also requires vector rotation so it also has a CORDIC on rotate mode. The CORDIC design is exactly the same from coarse frequency correction.

After having all receiver chain on BSV designed and validated against Python results, comparing both SQNR and graphically all inputs and outputs of each block, the limiter bit masking design started. BSV allows the programmer to manipulate integer and fractional parts of any Fixed-Point type as number or bit array so for each operand on BSV calculations, a bit-mask with the same FWL size was created and controlled. The masking operation (the act of masking the FWL bits of a given operand) happens before operation for input operands and after the operation for the result operand. As vectors are accessed in an iterative fashion, masks were applied during each index usage. For initial testing, masks full of 1s were applied and no difference noticed from before masking, which is the expected behavior.

3.5 Simulation strategies

For Python simulation with BSV results, the original Python algorithm was broken into pieces. Each piece goes from the creation of a packet until the result of the block under test. The steps for simulation are:

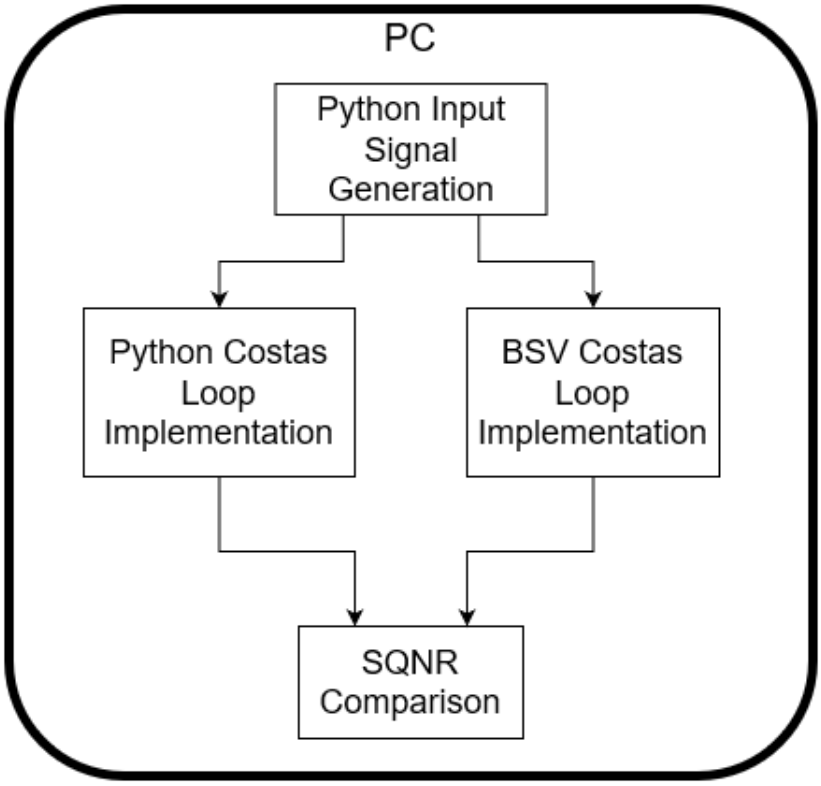

Figure 16 – Software simulation strategy.

- ❏ Python generates a data packet and applies all operations until the block under test stage.
- ❏ Python prints the number of bits masked to each operand being controlled and the complex values of the data packet on a log file.
- ❏ Python invokes BSV block for a simulation run on Verilator where the log file is parsed to control FWL bit-mask and provide data. BSV ends printing a log file of the transformation applied to input data.
- ❏ Python then performs the same transformation, reads the output data from BSV and calculates the SQNR between the outputs.

The items above can be exemplified as shown on Figure 16, with the steps for Costas Loop simulation.

These SQNR results were used on a max-1 algorithm to find the mask set where the minimal SQNR result is above 15 dB.

3.6 Coarse Frequency Estimation

The first reception block has to fix the frequency difference between the oscillators on transmitter and receptor. Its first step is to estimate the frequency offset – the Delay and

Multiply algorithm finds it in terms of sampling frequency (f_s) and is capable of detecting offsets from $-0.5f_s$ to $+0.5f_s$ (MENGALI, 1997). The correction block uses the offset found to create a fixing signal and then multiply it by the received signal.

The frequency offset is first computed as an accumulated error vector based on the current and the last sample computed and then its argument (adjusted by $\text{sps}/2\pi T_{\text{symbol}}$) becomes the frequency offset as sampling frequency offset. This accumulation requires a minimum of 32 samples to correctly detect the frequency offset value.

To find the vector argument, an arctangent calculation is required. Python3 has an `atan2` function as reference and a CORDIC algorithm was implemented on BSV as its fixed point counterpart. The optimization process for Delay and multiply algorithm has a total of 14 signals to optimize: 8 operation signals and 3 from each CORDIC algorithm.

3.7 Timing Estimation

After adjusting carrier differences, it's now possible to adjust the timing error between the exact sampling moment at transmission and reception. As the system uses different clock sources and no clock reference is directly transmitted to synchronize these clocks, this system learns from the signal the best moment to sample it. The Muller & Mueller algorithm performs its task at 1 sample/symbol (MENGALI, 1997).

The implementation does not depend on any extra algorithm like CORDIC, it's just a series of complex calculations performed based on current and last 2 samples. The output is a vector containing just the best samples for each symbol transmitted. As this design works with 4 samples per symbol, this algorithm is a special form of decimation.

3.8 Costas Loop

The final step executes a fine frequency tuning based on computing phase and frequency errors. It's a straightforward algorithm: the phase is calculated based on real and imaginary samples. For BPSK modulation, imaginary values are minimized to zero and real values receive the energy from imaginary. The frequency error is computed as a phase accumulation error, and phase itself is an instantaneous error. To perform the fix, CORDIC is used to rotate the sample back, removing the phase and frequency errors. This algorithm fixes sample by sample and does not require any sample buffering.

Chapter 4

Results and discussion

This chapter presents the simulation, optimization and synthesis results for each DSP block proposed as well as discussions about the data obtained on each step.

4.1 Coarse Frequency Estimation Simulation

A counter intuitive finding about Delay and multiply algorithm is it performs better with continuous signals. Transitions on the received signal amplifies the instability during accumulation more than a continuous signal. An example of this phenomena can be seen on Figures 17 and 18, where the imaginary samples starts growing in amplitude because of instability.

As the output varies according to the signal input because some of the instabilities presented on Menard et al. (2019a), each simulation step with a given set of bit-masks is done with 100 inputs, to allow the computation of mean and minimal SQNR values. The max-1 algorithm choose the limiting set which corresponds to the biggest minimal SQNR for the round.

All fixed-point signals from Delay and multiply started with 16 fractional bits for all operands, showing up to 40dB for SQNR. The result of max-1 was: `[13, 13, 13, 8, 12, 12, 12, 12]` bits masked for Delay and multiply itself, `[9, 9, 6]` for CORDIC rotate algorithm and `[16, 13, 5]` bits masked for CORDIC atan algorithm. The signals controlled were `[currentSample, lastSample, accumSample, fsError, input, output, xFix, yFix]` for D&M and `[x, y, z]` for CORDIC algorithms. The meaning of each signal can be found on the algorithm's code on author's GitHub. For D&M, the most sensible signal is fsError, with only 8 bits masked, compared to all other with 12 or more, it also reflected on CORDIC rotate algorithm, where fsError is z input (with 6

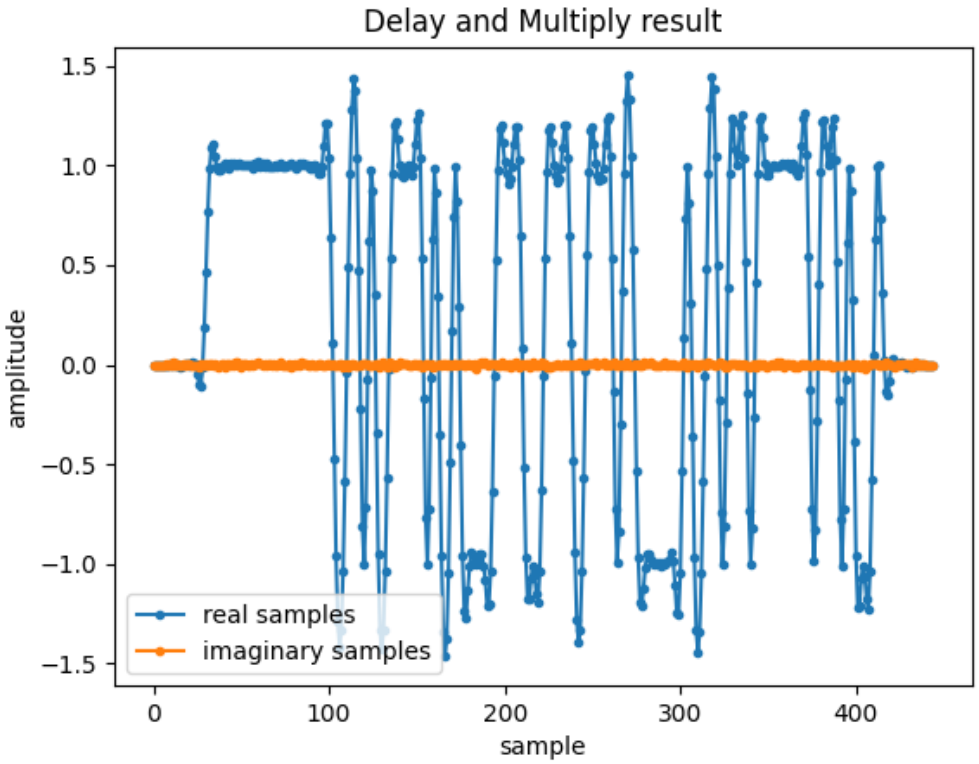

Figure 17 – Stable packet processing. Source: Author

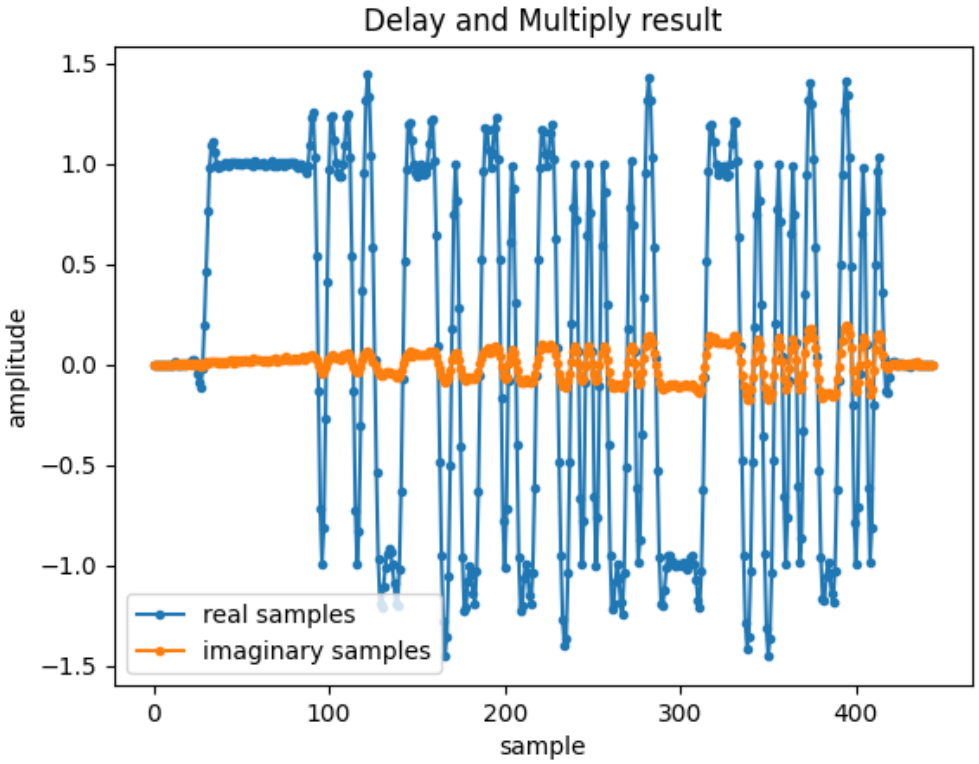

Figure 18 – Unstable packet processing. Source: Author

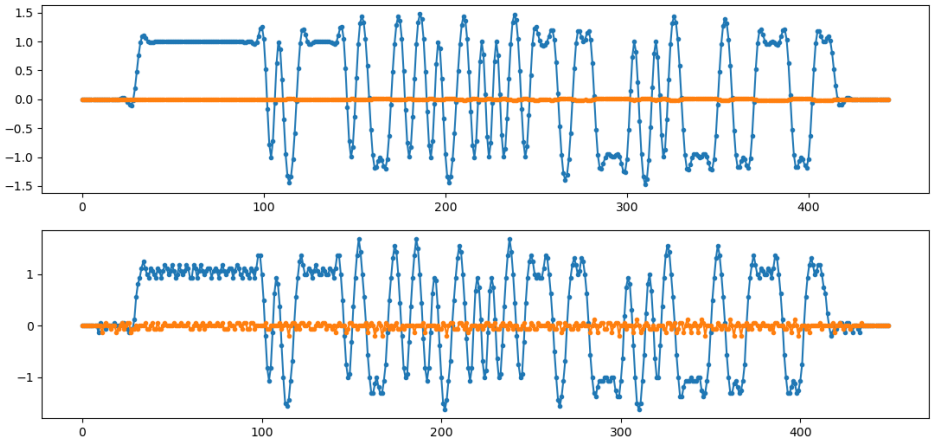

Figure 19 – D&M - Python to optimal result comparison. Source: Author

bits masked). An optimal CORDIC for D&M was designed so the synthesis result could show the better results. The Figure 19 shows the output of Python processing compared to the output of the optimal signal. The upper figure is the Python output and lower, BSV output. BSV produces an "unsmooth" signal since masking LSB bits diminishes the detailing of the signal contour. The time spent on max-1 to reach the result is 3h33min.

4.2 Timing Estimation Simulation

All fixed point operands were also configured to 16 bit on fractional and showed a stable minimal SQNR of approximately 19dB across many masks sets. The optimal values found by max-1 were `[15, 14, 12, 12, 14]` where the signals are `[x, y, mu, input, mmVal]`, all of them complex.

Since the BSV standard library currently doesn't have any functions to adapt complex values of different integer or fractional sizes (equivalent to the `fxptSignExtend` and `fxptTruncate` functions used with non-complex values), we chose the suboptimal mask `[12, 12, 12, 12, 12]` for easiness of implementation. As a future work, we intend to implement these functions, allowing to easily mix complex values of different sizes.

The time spent running max-1 to find the result was 19min38s. Figure 20 shows the outputs of Python (upper) and BSV (lower). There is some noise on imaginary samples (orange) but the Real samples (where the data resides for BPSK) have a similar shape.

4.3 Costas Loop Simulation

The implementation showed a very high minimal SQNR of 69dB starting with 12 bits for fixed-point fractional so the max-1 started the simulations from there. Max-1 result is

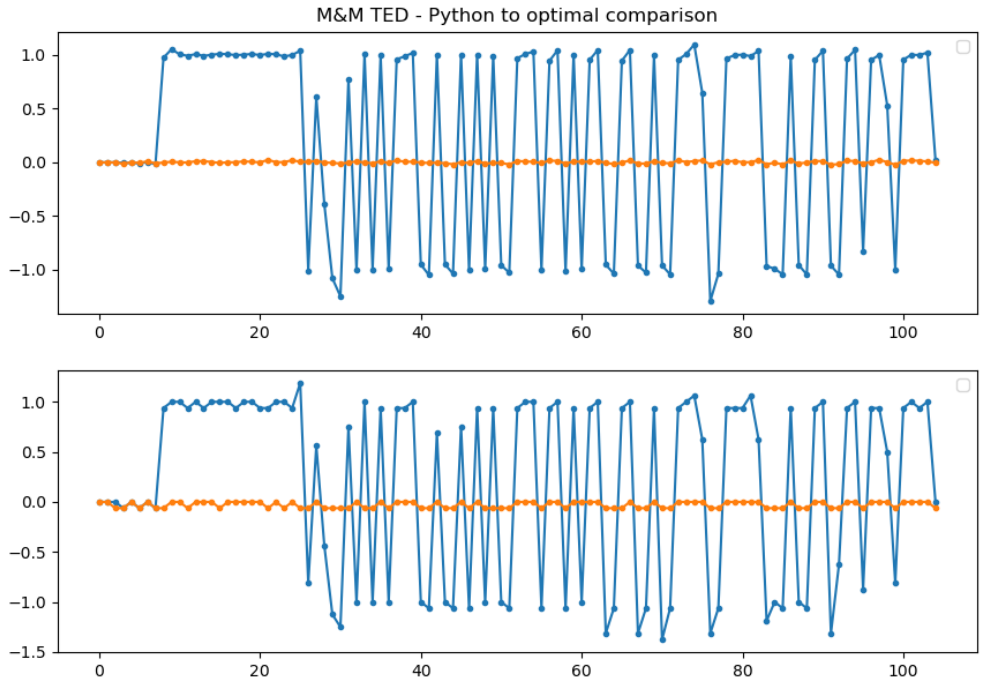

Figure 20 – M&M - Python to optimal result comparison. Source: Author

`[12, 13, 7, 12, 12]` for Costas Loop and `[10, 10, 11]` for CORDIC rotate algorithm. Costas loop signals are `[phase, frequency, error, input, output]` and CORDIC are `[x, y, z]` just as with D&M. An optimal CORDIC for Costas Loop was also prepared for synthesis. Figure 21 shows the outputs of Python (upper) and BSV (lower). It has a similar output to M&M but has more noise on imaginary (orange samples). At this stage, the transmitted data is on the blue samples – everything above 0 corresponds to 1. Comparing the blue samples, we can see every sample.

4.4 FPGA synthesis

The target hardware for synthesis is a Xilinx Spartan6 XC6SLX116 FPGA, with approximately 16000 logic elements and a low cost development board, which would allow to build a testbed IoT network once a RF front-end was developed. As some designs became bigger that the capability of the chosen hardware, for synthesis data collection, a bigger part number was used: XC6SLX100, with 100000 logic elements. More details about the FPGAs can be found on their datasheet (AMD, 2011).

For synthesis, the I/O of the DSP blocks were connected to a simple UART, which allows sending and receiving data in a similar fashion to Python simulation. It's just a driver instead of a log file, and did not occupy many LUTs on the design (88 slice registers

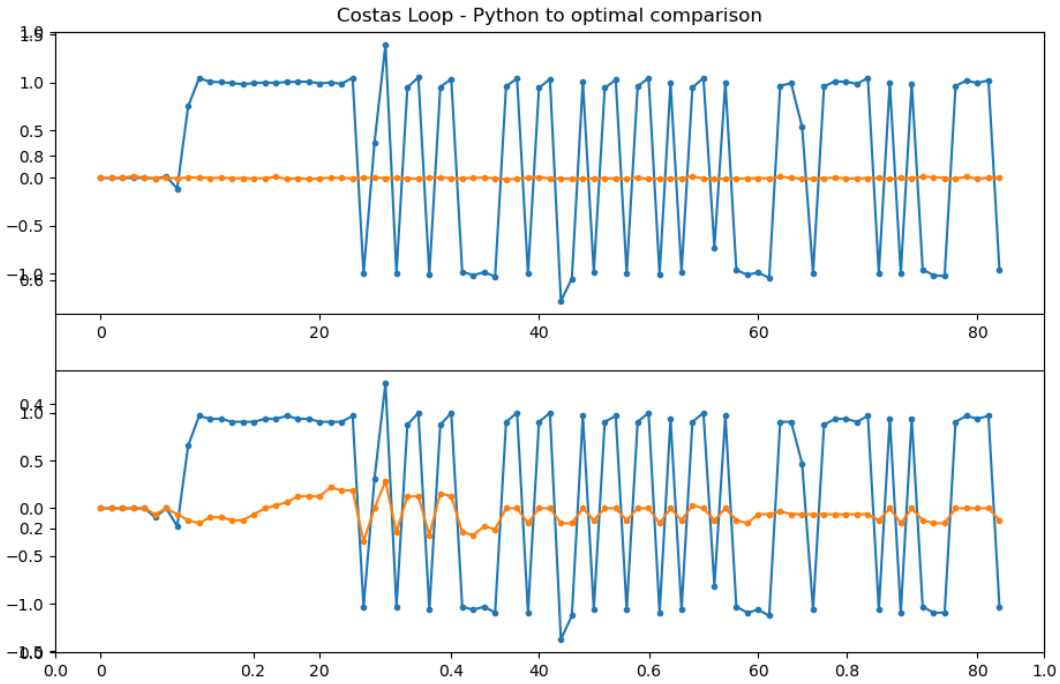

Figure 21 – Costas Loop - Python to optimal result comparison. Source: Author

and 234 slice LUTs), so the results are primarily related to their sizes and complexities.

Three designs were implemented for each block:

- ❏ a limited version, where the first bytes received by FPGA were the bit-masks, then samples;
- ❏ a non-limited version, just like the limited in terms of size of fractional words but no limiting operation and no limiting registers. Just the block with no optimization;
- ❏ the optimized block, using the best result from max-1 and all extra logic to truncate or sign extend fractional words between calculations.

The data transferred between FPGA and PC has a binary format, the 1st byte representing integer, 2nd and 3rd bytes representing fractional, with no separator between numbers. If the fractional has less than 9 bytes, the 3rd byte is transmitted all zeroes.

Looking at Tables 1, 2 and 3 we can see data for 3 synthesized systems: limited (with bit-masks limiting operators), the "all x bits" with no masks and no optimization and the optimal systems with the results presented in the previous sections. The expected gains were obtained:

- ❏ Higher clock, less usage of Slice LUTs and Slice REGs, and lower power consumption for all optimal cases, showing the benefits of optimizing the circuit;

Table 1 – Resources and clock for Coarse Frequency

	limited	all 16 bits	optimal
clock (MHz)	21.867	37.512	41.293
Slice LUTs	24221	8577	4312
Slice REGs	16408	15877	8060
Power(mW)	190	176	162

Table 2 – Resources and clock for Muller & Mueller

	limited	all 16 bits	optimal
clock(MHz)	37.800	36.884	62.192
Slice LUTs	27143	6411	3457
Slice REGs	14876	14628	6627
Power (mW)	214	173	169

Table 3 – Resources and clock for Costas Loop

	limited	all 12 bits	optimal
clock(MHz)	36.511	35.034	63.395
Slice LUTs	2661	2572	1385
Slice REGs	1179	1016	503
Power (mW)	54	49	46

- ❏ Slice LUTs and Slice REGs decreasing between limited and "all x bits" for all cases, which is expected as the limiting masks use resources to operate;

- ❏ A smaller difference between limited and "all x bits" on Slice REGs.

There is no clear calculation to preview beforehand how much additional resource will be consumed after adding a mask to a design block. Since the transceiver DSP blocks implement very different algorithms, the amount of logic optimization synthesis tools are able to carry out varies from block to block.

Regarding power analysis, it's noticeable that at least 40% of power consumption shown on Tables 1, 2 and 3 are due to On-Chip leakages. Considering the IoT focus of this work, a strategy to also optimize leakages is an important direction for future works.

It can also be seen that a suboptimal search algorithm such as Max-1 could find aggressive results for all three DSP blocks designed, with not many bits left for optimization, even a case with zero fractional bits for a signal. This leads to some questioning about:

- ❏ how much better could other algorithms perform, like simulated annealing or evolutionary?

- ❏ how much would it cost to achieve such better results?

- ❏ which types of DSP systems would require better algorithms?

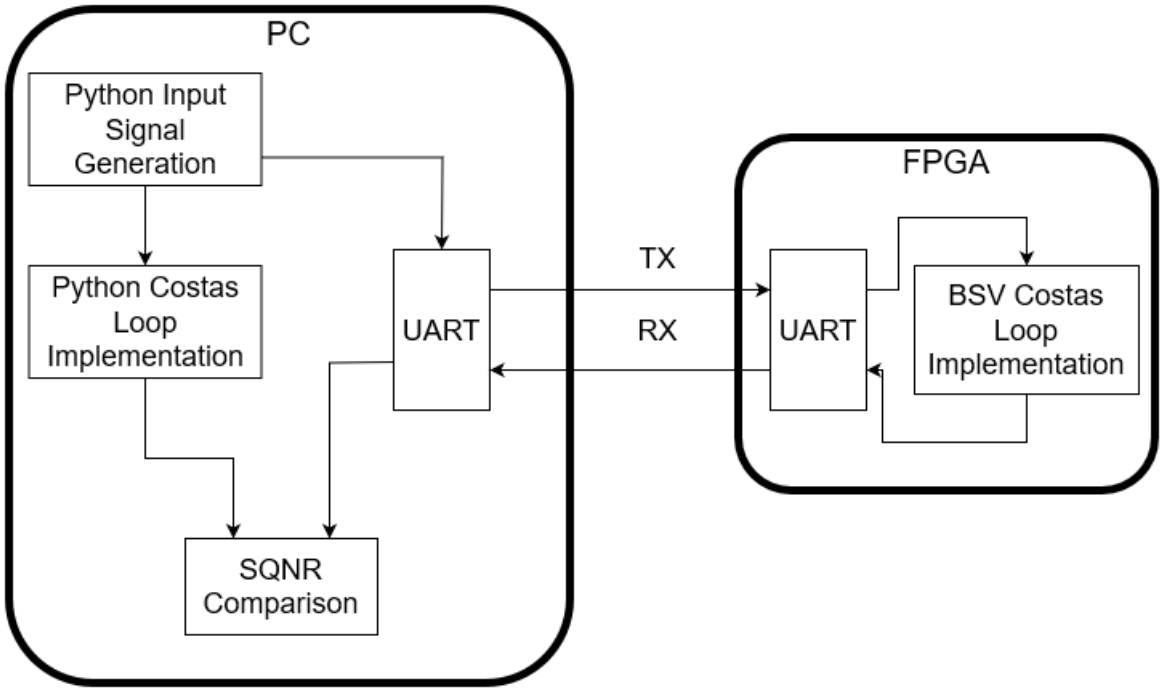

Figure 22 – Hardware-in-the-Loop simulation strategy.

4.5 Hardware-in-the-loop (HIL) simulation

The version of Costas Loop with bitmasking had a small size after synthesis, enough to fit on Xilinx XC6SLX16 FPGA, allowing to perform HIL simulations, so the scheme shown on Figure 22 was implemented. As the hardware design had a serial port, it was chosen as interface for HIL tests. The implemented version is a simple 8N1 9600 baud rate serial. The serial speed is very slow, approximately 1 byte per 1ms, so 3 bytes per sample on a 81 complex samples test leads to 500ms just transferring bytes to and from FPGA serial port for 1 input. A simulation has 100 input samples so the total time is 50s for one step of Max-1 algorithm. As a comparison, this result can be obtained in PC in 2-3s. The data flow is: FPGA starts sending 5 bytes to indicate start of operation, then the first N bytes sent by PC correspond to the mask values, each mask is a byte. Then PC starts sending the sample values as explained on section 4.4, where the real value goes first. As FPGA finishes the calculations, it send the results. Besides the very slow result, this test validates the feasibility of the idea proposed by Hormigo and Caffarena (2021) and the results obtained by HIL shows SQNR results above 50dB compared to the limited results obtained in software, which demonstrates that the HIL circuit is very similar to the software simulations so the results from HIL can be trusted and the “Static data formatting” step shown on Figure 1 can be merged with “Bit-true optimization” to reduce the time to design a chip.

Chapter 5

Conclusion

This work could demonstrate the usage of the technique proposed by Hormigo and Caffarena (2021) to perform bit-true simulations and optimize the fractional word length of operands on 3 blocks of the reception chain of a BPSK demodulator. The technique allows building the blocks integrated to bit masks, emulating math operations with limited precision with no resynthesis of the system. With these limitators in place, simulations in software and hardware were performed controlling the precision of each operation to find the best values according to the designer needs.

5.1 Future work

An extension of this work would be to improve the communication between HIL and PC, using faster buses such as PCI Express, USB or even SoCs like AMD's Zynq and Intel's Cyclone V where the chip has a high speed bus to perform communication between an integrated hard processor and synthesized logic. Another path of research is to compare different types of DSP systems to better understand how each search algorithm performs and better compare the gains of each search based on the penalties each technique also impose. A third path is to restart the work described in the "Of what went wrong" section and create the "LimitedVar" as part of BSV, Chisel, SpinalHDL or other HLS tool to finish the idea of having a tool where the block can be designed as production and the optimization process and simulations can be delivered as a compilation/synthesis option.

References

ALGARABEL, E. S. Automated wordlength optimization framework for multi-source statistical interval-based analysis of nonlinear systems with control-flow structures. 2016. Disponível em: <https://oa.upm.es/40614/>.

AMD. **Spartan-6 Family Overview**. [S.l.], 2011. Rev. 2.0, avaiable at <https://docs.amd.com/v/u/en-US/ds160>.

CANTIN, M.-A.; SAVARIA, Y.; LAVOIE, P. A comparison of automatic word length optimization procedures. In: **2002 IEEE International Symposium on Circuits and Systems. Proceedings (Cat. No.02CH37353)**. [S.l.: s.n.], 2002. v. 2, p. II–II.

CHERUBIN, S.; AGOSTA, G. Tools for reduced precision computation: A survey. **ACM Comput. Surv.**, Association for Computing Machinery, New York, NY, USA, v. 53, n. 2, apr 2020. ISSN 0360-0300. Disponível em: <https://doi.org/10.1145/3381039>.

CONSTANTINIDES, G.; CHEUNG, P.; LUK, W. Wordlength optimization for linear digital signal processing. **IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems**, v. 22, n. 10, p. 1432–1442, 2003.

FLEMING, K. E. et al. Wilis: Architectural modeling of wireless systems. In: **(IEEE ISPASS) IEEE International Symposium on Performance Analysis of Systems and Software**. [S.l.: s.n.], 2011. p. 197–206.

GREISEN, P.; HAENE, S.; BURG, A. Simulation and emulation of mimo wireless baseband transceivers. **Eurasip Journal On Wireless Communications And Networking**, p. 196796, 2010. Disponível em: <http://infoscience.epfl.ch/record/166572>.

HA, V.-P.; YUKI, T.; SENTIEYS, O. Towards generic and scalable word-length optimization. In: **Proceedings of the 23rd Conference on Design, Automation and Test in Europe**. San Jose, CA, USA: EDA Consortium, 2020. (DATE '20), p. 1668–1673. ISBN 9783981926347.

HAN, K.; EVANS, B. L. Optimum wordlength search using sensitivity information. **EURASIP Journal on Advances in Signal Processing**, Springer, v. 2006, p. 1–14, 2006.

HORMIGO, J.; CAFFARENA, G. Fpga acceleration of bit-true simulations for word-length optimization. In: **2021 IEEE 28th Symposium on Computer Arithmetic (ARITH)**. [S.l.: s.n.], 2021. p. 119–122.

KEDING, H. et al. Fridge: a fixed-point design and simulation environment. In: **Proceedings Design, Automation and Test in Europe**. [S.l.: s.n.], 1998. p. 429–435.

KUM, J. K. K.-I.; SUNG, W. Autoscaler for c: an optimizing floating-point to integer c program converter for fixed-point digital signal processors. **IEEE Transactions on Circuits and Systems II: Analog and Digital Signal Processing**, v. 47, n. 9, p. 840–848, 2000.

KUM, K.-I.; KANG, J.; SUNG, W. A floating-point to integer c converter with shift reduction for fixed-point digital signal processors. In: **1999 IEEE International Conference on Acoustics, Speech, and Signal Processing. Proceedings. ICASSP99 (Cat. No.99CH36258)**. [S.l.: s.n.], 1999. v. 4, p. 2163–2166.

MENARD, D. et al. Analysis of finite word-length effects in fixed-point systems. In: _____. **Handbook of Signal Processing Systems**. Springer International Publishing, 2019. p. 1063–1101. Disponível em: <https://doi.org/10.1007/978-3-319-91734-4_29>.

_____. Fixed-point refinement of digital signal processing systems. In: **Digitally Enhanced Mixed Signal Systems**. The Institution of Engineering and Technology, 2019. p. 1–37. Disponível em: <https://hal.inria.fr/hal-01941898>.

MENGALI, U. **Synchronization Techniques for Digital Receivers**. Springer US, 1997. (Applications of Communications Theory). ISBN 9780306457258. Disponível em: <https://books.google.com.br/books?id=89Gscsw7PvoC>.

NEHMEH, R. et al. Integer word-length optimization for fixed-point systems. In: **2014 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)**. [S.l.: s.n.], 2014. p. 8321–8325.

NG, M. C. Airblue : a highly-configurable fpga-based platform for wireless network research. 2011. Disponível em: <https://dspace.mit.edu/handle/1721.1/66457>.

OSBORNE, W. et al. Automatic accuracy-guaranteed bit-width optimization for fixed and floating-point systems. In: **2007 International Conference on Field Programmable Logic and Applications**. [S.l.: s.n.], 2007. p. 617–620.

ROSA, L. et al. Design and analysis of evolutionary bit-length optimization algorithms for floating to fixed-point conversion. **Applied Soft Computing**, v. 49, p. 447–461, 2016. ISSN 1568-4946. Disponível em: <https://www.sciencedirect.com/science/article/pii/S156849461630429X>.

SEMTECH CORPORATION. **SX1276/77/78/79 - 137 MHz to 1020 MHz Low Power Long Range Transceiver**. [S.l.], 2020. Rev. 7, avaiable at <https://www.semtech.com/products/wireless-rf/lora-connect/sx1276>.

SHAW. **The Legendary Fast Inverse Square Root**. 2017. <https://medium.com/hard-mode/the-legendary-fast-inverse-square-root-e51fee3b49d9>.

STMICROELECTRONICS. **Low data rate, low power Sub 1GHz transceiver**. [S.l.], 2021. Rev. 11, avaiable at <https://www.st.com/en/wireless-connectivity/spirit1.html>.

_____. **Ultra-low power, high performance, sub-1GHz transceiver**. [S.l.], 2023. Rev. 11, avaiable at <https://www.st.com/en/wireless-connectivity/s2-lp.html>.

WU, B.; ZHU, J.; NAJM, F. Dynamic-range estimation. **IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems**, v. 25, n. 9, p. 1618–1636, 2006.