FUNDAMENTAL OF DIGITAL SYSTEM FINAL PROJECT REPORT
DEPARTMENT OF ELECTRICAL ENGINEERING
UNIVERSITAS INDONESIA

QPSK Modulator Implemented with the Cordic Algorithm using FPGA

GROUP 5

Saddam Titanio Darmawan	**2406450472**
Lando Akmalkane Airell Muhammad	**2406450390**
Benintya Farrel Armaya	**2406450346**
Andika Mangaraja Nainggolan	**2406450333**

PREFACE

This final project represents the design, and implementation of a FPGA-based QPSK Modulator using the CORDIC algorithm. This work is carried out as a requirement for completing the Final Project for Digital System Design Course.

This FPGA is used to process a binary data stream, and process it into a phase-modulated carrier signal, which the signal is suitable for transmission. The input bitstream is grouped into pairs of bits, with each pair mapped to one of four possible points in the QPSK constellation diagram. Then, each constellation point corresponds to amplitude values of +1 or -1 for the in-phase (I), and quadrature (Q) components. The components (I, and Q) are multiplied by the orthogonal carrier signals, whic usually is cosine for the in-phase (I) path, and sine for quadrature (Q) path, thus the resulting signals are summed to produce the final output.

Since it uses sine, and cosine values. To efficiently generate the required values, we use the CORDIC (Coordinate Rotation Digital Computer) algorithm, which allows trigonometric computations to be implemented using shift, and add operations.

Depok, December 7th, 2025

Group 5

TABLE OF CONTENTS

CHAPTER 1

INTRODUCTION

1.1 BACKGROUND

Quadrature Phase Shift Keying (QPSK) Modulation is a digital modulation technique that is commonly used in wireless communications, such as WLANs/Wi-Fi and WiMAX standards.

1.2 PROJECT DESCRIPTION

This project implements the full QPSK modulation process by grouping incoming bits into symbol pairs and mapping them to their corresponding locations in the QPSK constellation diagram, where each symbol represents a unique phase shift of the carrier signal. The in-phase (I) and quadrature (Q) components are assigned values of +1 or –1, then multiplied by orthogonal carrier signals, cosine for the I path and sine for the Q path. These carrier signals are generated using a hardware-efficient CORDIC algorithm, which replaces multiplications with iterative shifts, add, and subtract operations to compute sine and cosine values. A phase accumulator drives the CORDIC module, and the final modulated waveform is produced by summing the I cos and Q sin components. The complete design is implemented in VHDL, simulated, and validated to demonstrate a working FPGA-based QPSK modulator.

1.3 OBJECTIVES

The objectives of this project are as follows:

1. Design and implement an FPGA-based QPSK modulation system.
2. Convert an incoming binary data stream into a QPSK-modulated waveform.
3. Implement symbol mapping to translate bit-pairs into I/Q constellation points.

4. Use a phase accumulator to generate continuous phase values.

5. Apply the CORDIC algorithm to efficiently compute sine and cosine carrier signals.
6. Mix the I and Q components with their respective carriers to form the modulated output signal.

1.4 ROLES AND RESPONSIBILITIES

The roles and responsibilities assigned to the group members are as follows:

Roles	Responsibilities	Person
Role 1	- Creating the Python script to generate the micro-rotation angles in Hexadecimal format - Researching and implementing the CORDIC algorithm - Creating the top-level entity that connects all of the modules	Saddam
Role 2	- Implement the IQ mixer - Implement the phase accumulator - Making the Preface	Lando

Role 3	- implement QSPK symbol mapping block - explained the objective and description of project	Ben
Role 4	- implement full adder block - implement the n bit adder block	Aal

Table 1. Roles and Responsibilities

CHAPTER 2

IMPLEMENTATION

2.1 EQUIPMENT

The tools that are going to be used in this project are as follows:

- Vivado 2025.1
- Python 3

2.2 IMPLEMENTATION

A QPSK modulator works by mapping the pairs of bits into one of the four possible symbols in the constellation diagram, as shown in Fig. 1. The purpose is to differentiate the pairs by applying different phase shifts to the carrier wave depending on the bits sequence. That way, the receiver can look at the phase of the incoming signal and perform the correct decoding method to recover the original two bits. Typically, bits 00, 10, 11, and 10 will have a phase shift of 0°, 90°, 180°, and 270° respectively. The first bit of the symbol determines the in-phase component, while the second bit determines the quadrature component. For instance, if the two bits were 00, the resulting symbol would be $1 + 1j$.

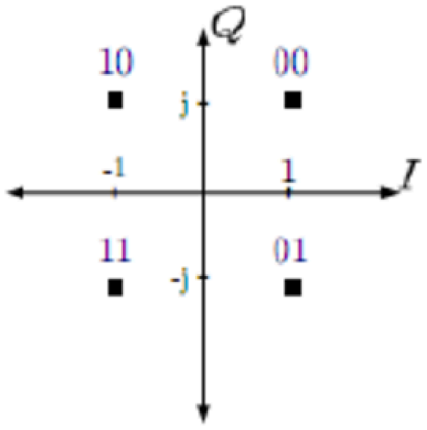


Fig 1. QPSK Constellation Diagram

The diagram of the process is shown in Fig 2. The outputs use two bits since the MSB will be used as the sign bit.

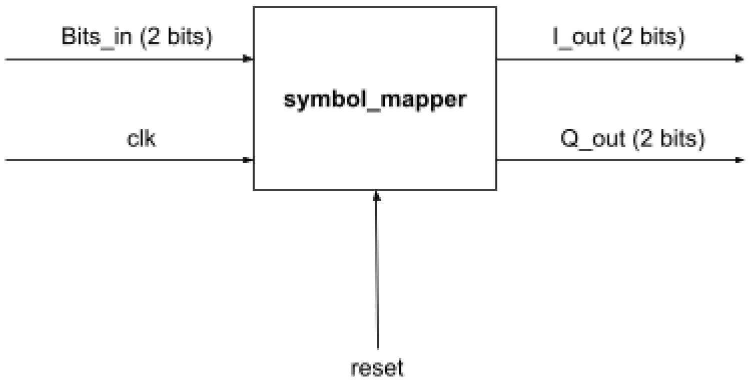


Fig 2. Black-box diagram of the IQ mapper

These components are then multiplied by their respective carrier signals. By convention, the I signal is modulated with a cosine waveform, and the Q signal is modulated with a sine waveform. To generate the carrier signals, a phase accumulator will increment the phase by a fixed amount each clock cycle and feed the resulting phase angle into a CORDIC algorithm to produce the corresponding sine and cosine waveforms. The phase accumulator will count from 0 to 2^N - 1 (mod 2^N), where N is the width of the register.

The CORDIC algorithm fundamentally operates through a sequence of vector rotations. It rotates a vector with an initial magnitude by a specified angle using incremental micro-rotations. The accuracy of the CORDIC result increases with the number of iterations since each additional iteration tunes the rotation by a smaller micro-angle. The algorithm is derived from the general rotation transform equations:

$$x' = xcos(\alpha) - ysin(\alpha)$$

$$y' = xsin(\alpha) + ycos(\alpha)$$

These can be rearranged such that

$$x' = cos(\alpha)(x - ytan(\alpha))$$

$$y' = cos(\alpha)(y + xtan(\alpha))$$

If the angles are chosen such that

$$tan(\alpha) = 2^{-i} \text{and therefore } \alpha_i = arctan(2^{-i}),$$

then the multiplication by the tangent term can be implemented using a right-shift operation since division by 2^n in binary corresponds to shifting right by n bits. However, the computational process has not yet been entirely simplified since it still involves a cosine multiplication at each iteration.

To resolve this issue, the cosine multiplication can actually be factored out of the iterative process as long as we apply a "correction factor" at the very end. This is necessary because each iteration stretches the length of the vector and we need a correction factor called gain to revert the vector to its initial length. At iteration i, the vector gets scaled by $1/cos(arctan(2^{-i}))$. Thus the correction is given by:

$$A = \prod_{i-0}^{n-1} \frac{1}{cos(arctan(2^{-i}))} = \prod_{i-0}^{n-1} \sqrt{1 + 2^{-i}}$$

$$A = \lim_{n \to \infty} \left(\prod_{i-0}^{n-1} \sqrt{1 + 2^{-i}} \right) \approx 1.64676$$

The output of the CORDIC algorithm will be a signed 16-bit value. Thus, the gain that will be

$$GAIN = \frac{2^{15}-1}{1.64676} \approx 0x4DB7$$

Now that the cosine multiplications are no longer included during the iterative process, the CORDIC algorithm only involves shift, addition, and subtraction operations.

$$x_{i+1} = x_i - y_i 2^{-i}$$

$$y_{i+1} = y_i + x_i 2^{-i}$$

The CORDIC micro-rotation angles are generated using a Python program. The set of angles will function as a lookup table that will be referred to for each iteration. Each angle is converted from radians to degrees and then scaled into 32-bit phase-accumulator units. It will print each value in the appropriate hexadecimal format so that it can be directly pasted into the VHDL script.

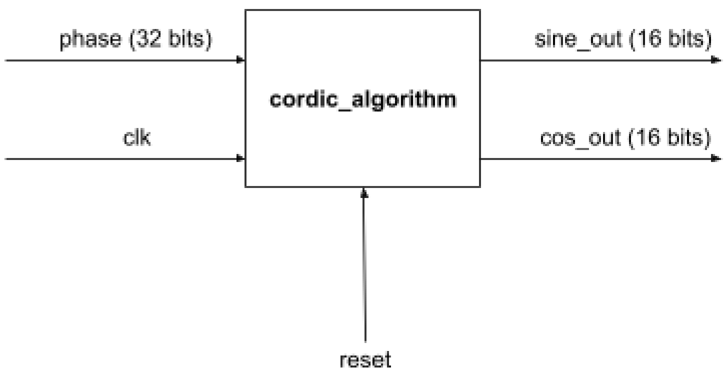


Fig 3. Black-box Diagram of the Cordic Algorithm

Once the cosine and sine carrier signals are generated, they will be multiplied by the value of the I and Q components, respectively. The resulting output will be 18 bits to prevent overflow since a multiplication between a 16-bit and a 2-bit value will at most yield an 18-bit value. Afterward, the two results are added to produce the final modulated signal. An n-bit adder is used to achieve this functionality, which consists of full adders generated using the for generate construct in VHDL. Finally, structural modeling is employed to interconnect all the lower-level components to produce the modulated signal based on the bit-pairs.

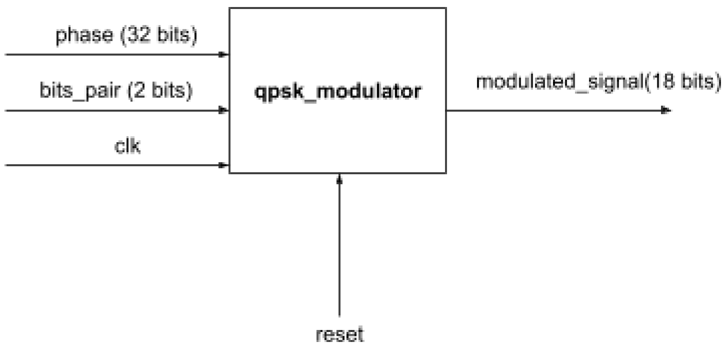


Fig 4. Black-box Diagram of the Top-Level Entity

CHAPTER 3

TESTING AND ANALYSIS

3.1 TESTING

The testing objectives and general process of each module can be summarized by the following table:

Module	Test Objective	Methodology
QPSK Mapper	The mapper correctly outputs the constellation symbol for every valid input bit-pair.	● Provide all four possible bit-pair inputs: 00, 01, 10, 11 ● Check that the output matches the expected I/Q symbol mapping.
CORDIC Sine/Cosine Generator	The CORDIC algorithm produces reasonably accurate sine and cosine values for the given phase input.	● Use a range of values within the 360° phase range ● A Python script will be used to compute the RMS error of the cosine and sine values obtained using the CORDIC algorithm
Top-Level QPSK Modulator	Verify that the top-level integration of symbol mapping, phase accumulation, CORDIC carrier generation, and IQ mixing into a final modulated signal.	● Provide all four possible bit-pair inputs of 00, 01, 10, 11. ● Observe the final modulated waveform and confirm that each bit pair produces the correct phase-shifted carrier

Table 2. Testing Objectives and Methodologies

3.2 RESULT

The results are conducted by creating testbenches that comply with the methodologies that have been established for each module.

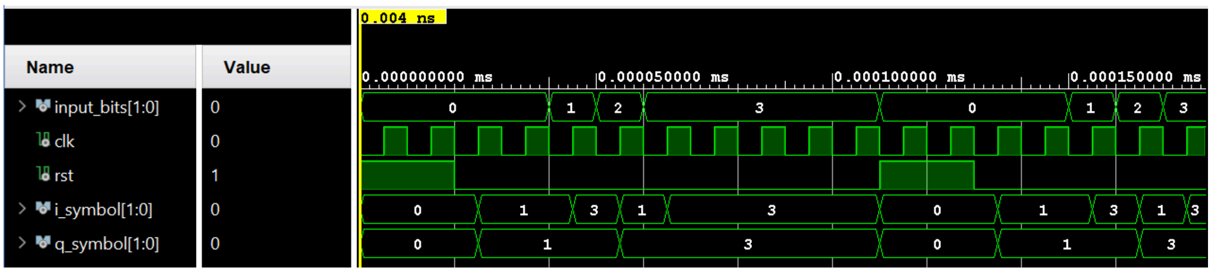


Fig 5. Waveform of the QPSK Mapper

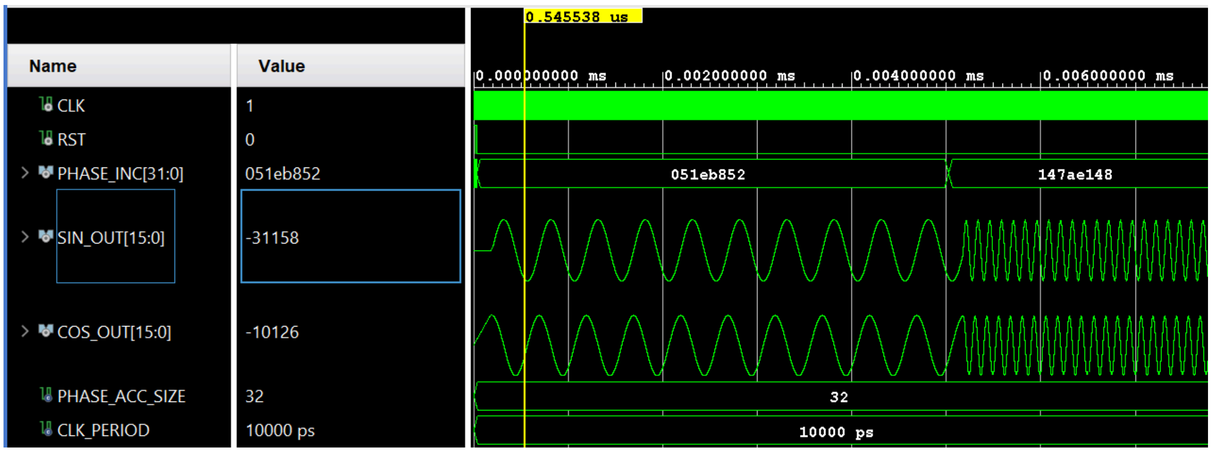
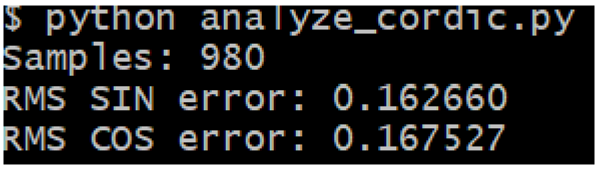


Fig 6. Waveform of the CORDIC Sine/Cosine Generator



```
$ python analyze_cordic.py
Samples: 980
RMS SIN error: 0.162660
RMS COS error: 0.167527
```

Fig 7. Screenshot of the RMS Error of the Sine and Cosine values

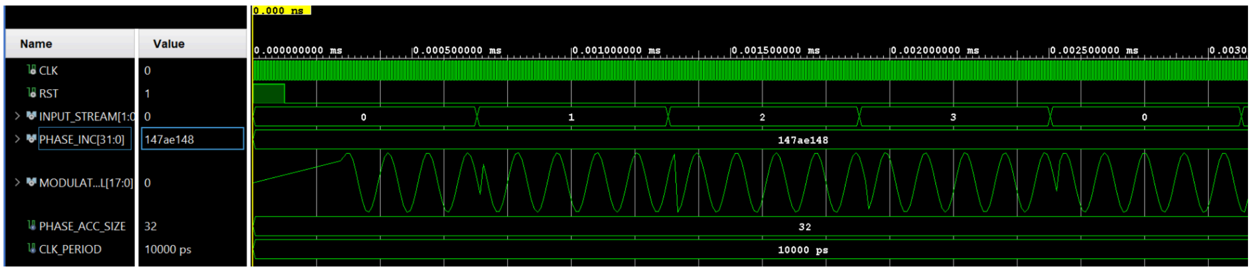


Fig 8. Waveform of the QPSK Modulator

3.3 ANALYSIS

The QPSK Mapper correctly outputs the symbols based on the bit-pair combinations. For instance, when the input_bits are both zero, the symbol for both the I and Q components are 1, as intended; when the input_bits is 01, the resulting symbol for the I component is 3 (since it is signed) and 1 for the q component.

For the CORDIC sine/cosine generator, the waveform displays the resulting sine and cosine waves with different carrier frequencies. Visually speaking, the waveforms are properly generated since the 90 degree phase difference between the two signals are apparent. This quadrature property between the two indicates that the design can support QPSK modulation. Additionally, the sine and cosine waves with the higher frequency have shorter wavelengths. However, RMS error for the sine and cosine values are approximately 0.16 - 0.17. These values provide the average difference between the experimental values (the ones generated with the implemented CORDIC algorithm) and the actual values. This indicates that the implementation of the CORDIC algorithm is not yet optimal and requires improvements - potentially by extending the width of the output, increasing the number of iterations performed, adjusting the gain value, and reviewing the sampling method.

Lastly, the simulation results show that each module within the QPSK modulation chain operates correctly. When combined in the top-level design, the I/Q mixer correctly multiplies the mapped symbols with their respective carriers, and the final output waveform exhibits the expected QPSK characteristics, including four distinct phase states. The simulation waveforms confirm that the modulated output follows the correct phase shifts corresponding to the input data stream. When the bit pair changes, the vertical boundary shows the carrier suddenly changing phase.

CHAPTER 4

CONCLUSION

This project demonstrates an FPGA-based QPSK modulator which uses CORDIC algorithm to efficiently generate the sine, and cosine carrier signals. The QPSK mapper converts every input bit-pair into the intended I/Q symbols, the CORDIC block produces quadrature signals (sine and cosine waves), next the integrated top-level design gives a modulated output with the expected four phase states. The measured RMS errors of about 0.16 from Fig 7. shows that CORDIC implementation is reasonably accurate but still far from being optimal. But overall, the main objective of this project, which is building a working QPSK modulator using CORDIC algorithm is a success.

REFERENCES

[1] M. Jain et al., “Performance optimized digital QPSK modulator,” 2017 International Conference on Energy, Communication, Data Analytics and Soft Computing (ICECDS), Chennai, India, 2017, pp. 68–71, doi: 10.1109/ICECDS.2017.8389540.

[2] “23.09: Phase Shift Keying Modulation,” Physics LibreTexts, Kettering University — Electricity and Magnetism with Applications to Amateur Radio and Wireless Technology. [Online]. Available: https://phys.libretexts.org/Courses/Kettering_University/Electricity_and_Magnetism_with_Applications_to_Amateur_Radio_and_Wireless_Technology/23%3A_Signal_Modulation/23.09%3A_Phase_Shift_Keying_Modulation

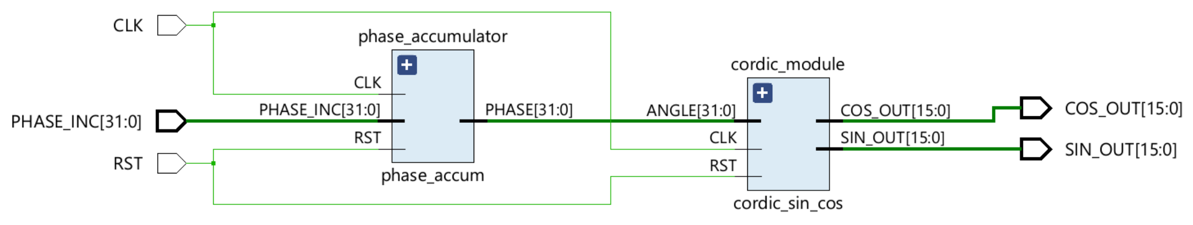
[3] Z. Z. Zhou, “Phase-noise, IQ imbalance and non-idealities in RF transceivers,” M.S. thesis, Dept. of Elect. Eng., Univ. of Padova, Padova, Italy, 2020. [Online]. Available: https://thesis.unipd.it/retrieve/71db069c-f6e5-4017-987f-6eb7f838a169/Zhou_Zhilong.pdf

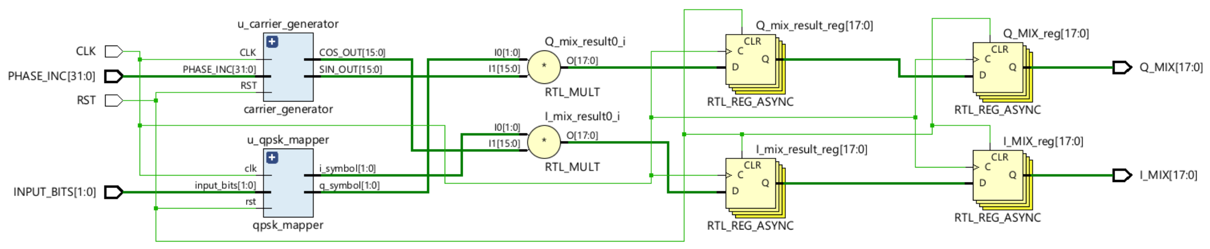
[4] D. Andraka, “CORDIC survey: algorithms, architectures and applications,” Xilinx, Inc., Mountain View, CA, USA, Tech. Rep., 1998. [Online]. Available: http://www.andraka.com/files/crdcsrvy.pdf

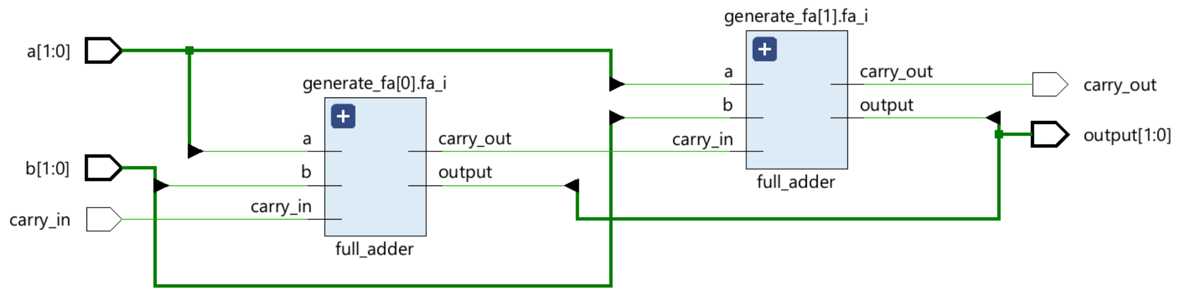
[5] R. K. Sakran, R. A. Zubaid, and A. A. Ali, “Design and implementation of QPSK modulator using FPGA based on DDS algorithm,” University of Thi-Qar Journal for Engineering Sciences, vol. 15, no. 1, 2024, doi: 10.31663/utjes.15.1.678.

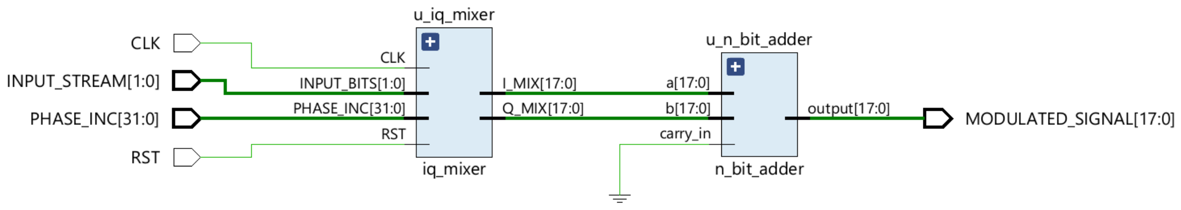
APPENDICES

Appendix A: Project Schematic



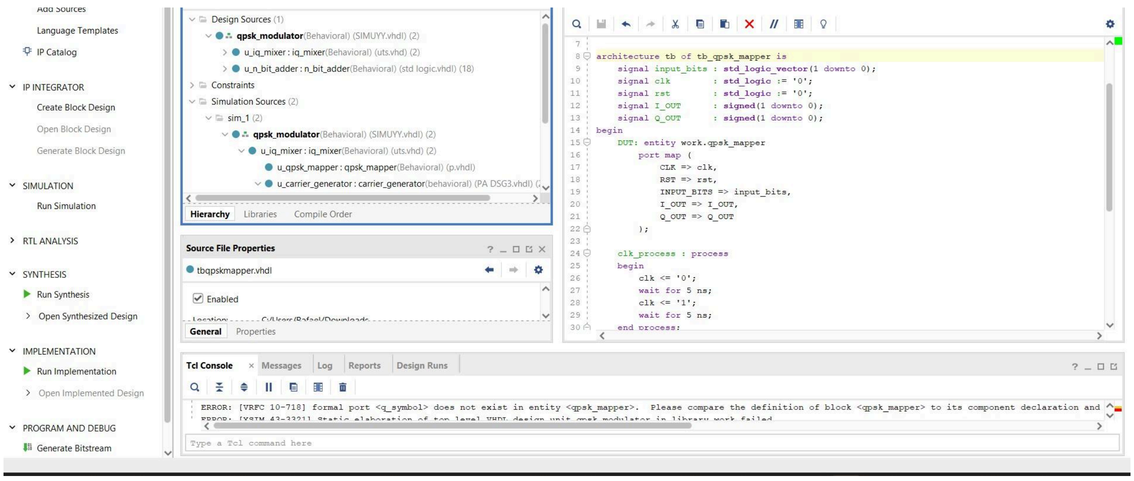






Appendix B: Documentation

Put the documentation (photos) during the making of the project



```
Design Sources (1)
  qpsk_modulator(Behavioral) (SIMUYY.vhdl) (2)
    u_iq_mixer : iq_mixer(Behavioral) (uts.vhd) (2)
    u_n_bit_adder : n_bit_adder(Behavioral) (std logic.vhdl) (18)
Constraints
Simulation Sources (2)
  sim_1 (2)
    qpsk_modulator(Behavioral) (SIMUYY.vhdl) (2)
      u_iq_mixer : iq_mixer(Behavioral) (uts.vhd) (2)
        u_qpsk_mapper : qpsk_mapper(Behavioral) (p.vhdl)
        u_carrier_generator : carrier_generator(behavioral) (PA DSG3.vhdl) (
```

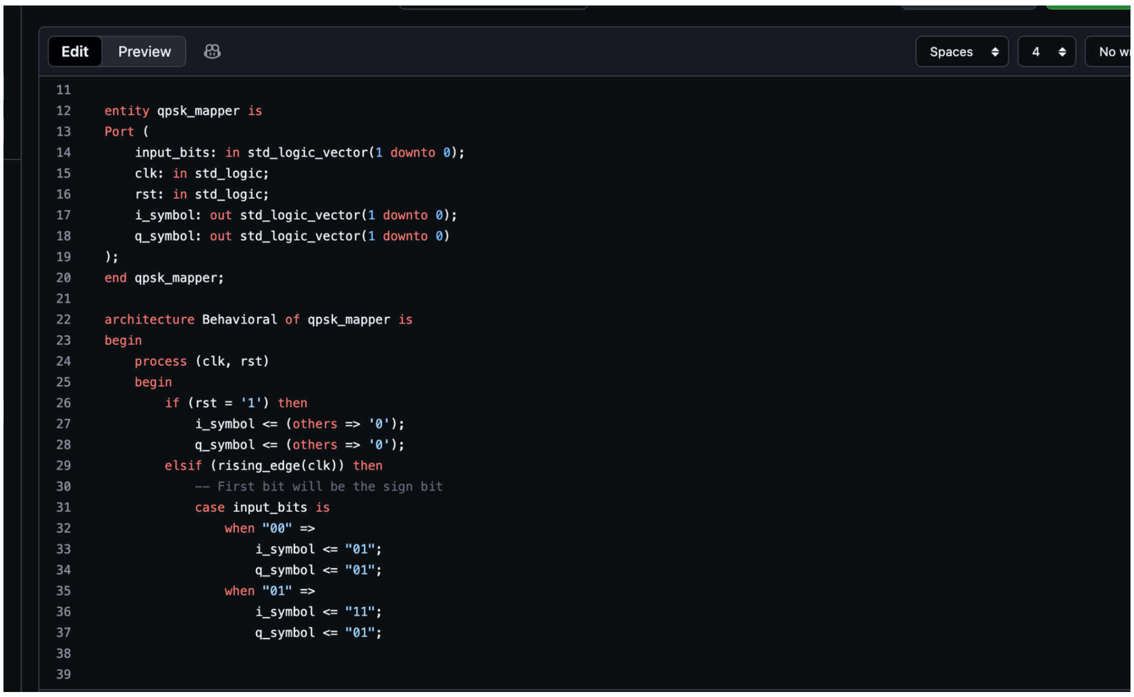
tbqpskmapper.vhdl

Enabled [✓]

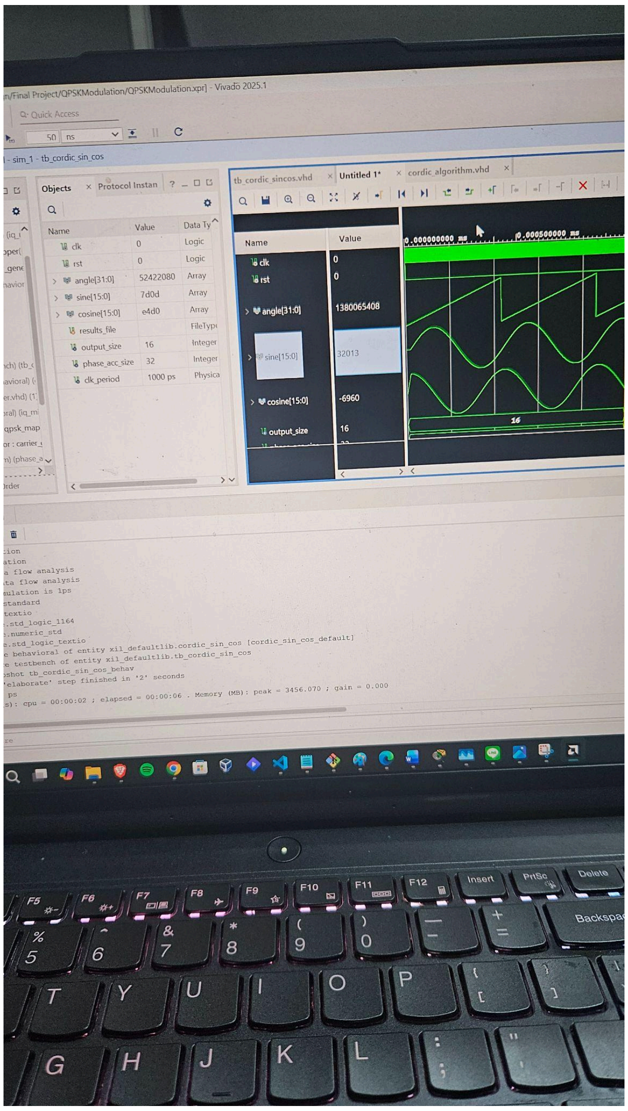
Location: C:/Users/Rafael/Downloads

```
architecture tb of tb_qpsk_mapper is
    signal input_bits : std_logic_vector(1 downto 0);
    signal clk        : std_logic := '0';
    signal rst        : std_logic := '0';
    signal I_OUT      : signed(1 downto 0);
    signal Q_OUT      : signed(1 downto 0);
begin
    DUT: entity work.qpsk_mapper
        port map (
            CLK => clk,
            RST => rst,
            INPUT_BITS => input_bits,
            I_OUT => I_OUT,
            Q_OUT => Q_OUT
        );

    clk_process : process
    begin
        clk <= '0';
        wait for 5 ns;
        clk <= '1';
        wait for 5 ns;
    end process;
```

```
ERROR: [VRFC 10-718] formal port <q_symbol> does not exist in entity <qpsk_mapper>.  Please compare the definition of block <qpsk_mapper> to its component declaration and
ERROR: [XSIM 43-3321] Static elaboration of top level VHDL design unit qpsk_modulator in library work failed.
```



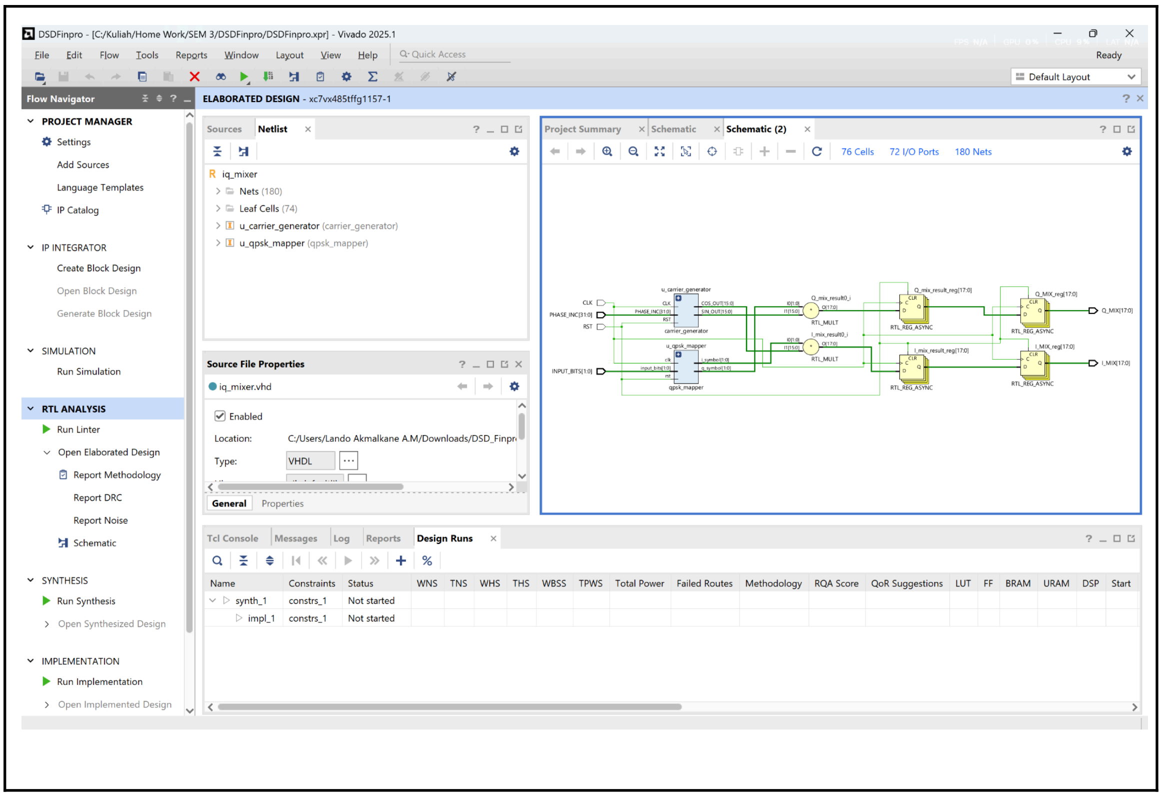
```

entity qpsk_mapper is
Port (
    input_bits: in std_logic_vector(1 downto 0);
    clk: in std_logic;
    rst: in std_logic;
    i_symbol: out std_logic_vector(1 downto 0);
    q_symbol: out std_logic_vector(1 downto 0)
);
end qpsk_mapper;

architecture Behavioral of qpsk_mapper is
begin
    process (clk, rst)
    begin
        if (rst = '1') then
            i_symbol <= (others => '0');
            q_symbol <= (others => '0');
        elsif (rising_edge(clk)) then
            -- First bit will be the sign bit
            case input_bits is
                when "00" =>
                    i_symbol <= "01";
                    q_symbol <= "01";
                when "01" =>
                    i_symbol <= "11";
                    q_symbol <= "01";


```



Name	Value	Data Ty
clk	0	Logic
rst	0	Logic
angle[31:0]	52422080	Array
sine[15:0]	7d0d	Array
cosine[15:0]	e4d0	Array
results_file		FileType
output_size	16	Integer
phase_acc_size	32	Integer
clk_period	1000 ps	Physica

Name	Value
clk	0
rst	0
angle[31:0]	1380065408
sine[15:0]	32013
cosine[15:0]	-6960
output_size	16

```
:ion
ation
a flow analysis
ita flow analysis
mulation is 1ps
standard
textio
:.std_logic_1164
e.numeric_std
e.std_logic_textio
e behavioral of entity xil_defaultlib.cordic_sin_cos [cordic_sin_cos_default]
e testbench of entity xil_defaultlib.tb_cordic_sin_cos
oshot tb_cordic_sin_cos_behav
'elaborate' step finished in '2' seconds
 ps
s): cpu = 00:00:02 ; elapsed = 00:00:06 . Memory (MB): peak = 3456.070 ; gain = 0.000
```



DSDFinpro - [C:/Kuliah/Home Work/SEM 3/DSDFinpro/DSDFinpro.xpr] - Vivado 2025.1

ELABORATED DESIGN - xc7vx485tffg1157-1

iq_mixer
- Nets (180)
- Leaf Cells (74)
- u_carrier_generator (carrier_generator)
- u_qpsk_mapper (qpsk_mapper)

Source File Properties

iq_mixer.vhd

[✓] Enabled

Location: C:/Users/Lando Akmalkane A.M/Downloads/DSD_Finpr

Type: VHDL

Name	Constraints	Status	WNS	TNS	WHS	THS	WBSS	TPWS	Total Power	Failed Routes	Methodology	RQA Score	QoR Suggestions	LUT	FF	BRAM	URAM	DSP	Start
synth_1	constrs_1	Not started																	
impl_1	constrs_1	Not started																	