UNIVERSITÀ DI PISA

Electronics and Communications Systems

Multi-Standard Modulator

Federico Cavedoni

Academic Year
2023/2024

Index

1.Introduction 4
Circuit Description 4
Possible Applications 4
Possible Architectures 5
2.Architecture description 6
3.VHDL Code 7
Multi Standard Modulator 7
Counter 8
Phase Adder 9
lut_table_65536_7bit 10
Amplitude Multiplier 10
QLUT Optimization 11
4.Test Phase 12
Frequecy modulation 12
Phase modulation 13
Amplitude modulation 13
Reset phase 13
QLUT Comparison 14
5.Vivado report 15
Elaborated design 15
Implementation 15
Timing and Critical path 15
Utilization 16
Power 16
Messages and warnings 17
6.Conclusion 18

1. Introduction

Circuit Description

This project presents the design and implementation of a digital Multi-Standard Modulator (MSM) capable of performing Amplitude Shift Keying (ASK), Frequency Shift Keying (FSK), and Phase Shift Keying (PSK) modulations.

The system employs a Look-Up Table (LUT) based on a Numerically Controlled Oscillator (NCO) that is able to generate the required waveforms for modulation. The NCO uses a precision phase accumulator to achieve fine frequency control and efficient signal synthesis.

The modulator circuit accepts input parameters for frequency, phase, and amplitude, enabling flexible configuration for various modulation schemes. These parameters are encoded as 16-bit values for frequency and phase, and as a 4-bit value for amplitude. The output of the modulator is a 16-bit digital signal representing the modulated waveform.

Possible Applications

The designed multi-standard modulator is applicable in diverse communication systems requiring flexible digital modulation:

- **Wireless Communication**: Supports QAM in LTE and 5G, PSK in Wi-Fi, and GFSK in Bluetooth, enhancing data rates and spectral efficiency.
- **Data Transmission**: Enables QAM in DSL and cable modems, and PAM in optical networks, ensuring high-speed data transfer.
- **Signal Processing**: Used in Software-Defined Radios (SDRs) for dynamic modulation adaptation with FSK, PSK, and QAM, and in Cognitive Radios for efficient spectrum use.
- **IoT Devices**: Supports ASK and PAM for efficient communication in smart home and LPWAN devices like LoRa and NB-IoT.
- **Satellite Communications**: Facilitates PSK and QAM in DVB for broadcasting and GNSS for reliable data transmission over long distances.

Possible Architectures

The multi-standard modulator will be composed of various components that will allow you to perform the required function:

- The **Numerically controlled Oscillator** (NCO) which will generate a wave based on an input frequency
- The **Phase Adder** which will take care of adding the input phase to the generated wave
- The **Look-up Table** (LUT) that allows you to obtain the value of the resulting wave with a simple lookup
- The **Amplitude Multiplier** which multiplies the signal by the input amplitude

These components allow to generate a wave and modulate it correctly based on the input frequency, phase and amplitude to obtain the modulated signal and use it for different types of communication and different protocols, as we have seen in the possible application paragraph.

2. Architecture description

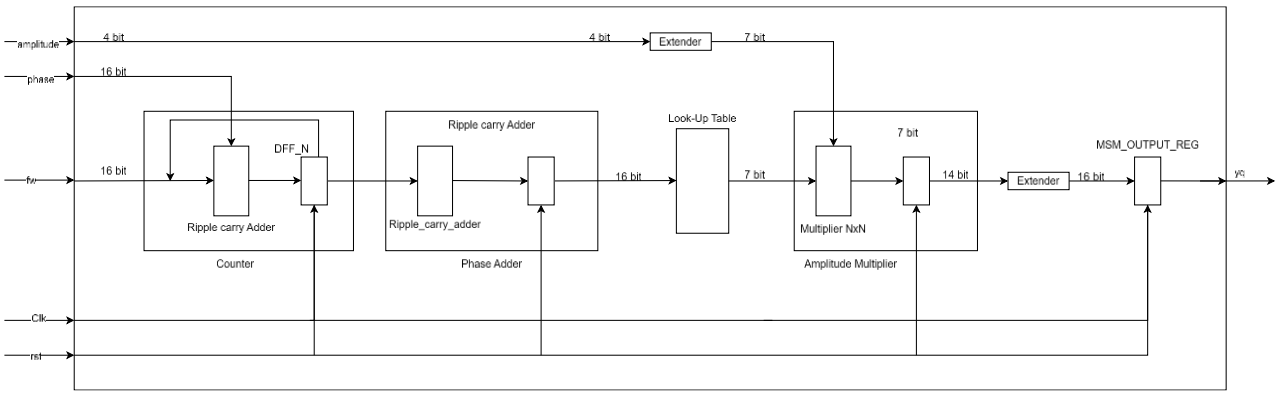

Figure 1: Architecture of the Multi-Standard Modulator

The system architecture, as introduced previously, is made up of various components that allow the correct application of the modulation functions.
As input to the system, we have the frequency word fw on 16 bits, the phase on 16 bits, the amplitude on 4 bits and the clock and reset signals. In output to the system, we only have the signal generated yq on 16 bits.

- The first block we find is the Counter which takes the clock, the reset and the frequency word as input and is responsible for generating the wave starting from the frequency multiplied by the k value of the counter. It returns as output k*fw on 16 bits.
- After the Counter there is the Phase Adder which takes care of adding the input phase on 16 bits to the k*fw frequency. In output I will therefore have k*fw + phase which is the index to be sent as input to the look-up table.
- After the phase adder there is the look-up table (LUT) which is a pre-compiled table with fixed input-output association. The table has a 16-bit input and a 7-bit output and returns a result that depends on the input index.
- After the lookup table there is the Amplitude Multiplier which takes as input the result of the lookup of the look-up table on 7 bits and the amplitude on 7 bits (4 originally, extended to 7 for the operation) and returns as output the signal multiplied by the input amplitude on 14 bits (7+7 from the multiplication).
- Finally, there is the MSM Output Register on 16 bits (14 originally from the multiplier extended to 16) which will output the modulated wave with the parameters passed as input.

3. VHDL Code

This section analyzes the various vhdl files that make up the system

Multi Standard Modulator

The MSM.vhd file is the main entity of the project and contains the declarations of the various subblocks such as counter and links starting from inputs to output.

```
entity MSM is
  generic (
    N : natural := 16;
    A : natural := 4;
    P : natural := 7;
    O : natural := 16
  );
  port(
    clk   : in std_logic;  -- clock of the system
    reset : in std_logic;  -- Asynchronous reset - active high

    fw : in std_logic_vector(N-1 downto 0);  -- input frequency word
    phase: in std_logic_vector(N-1 downto 0);  -- input phase
    amplitude : in std_logic_vector(A-1 downto 0);  -- input amplitude

    yq : out std_logic_vector(O-1 downto 0)   -- output waveform
  );
end entity;
```

Figure 2: MSM.vhd module

This module also takes care of extending the signals when necessary, as in the case of the amplitude input to the multiplier and the multiplier output to the system output.

```
  amp_ext <= (P-1 downto A => '0') & amplitude;
```

Figure 3: Amplitude bit extension

```
mul_ext <= (O-1 downto 2*P => multiplier_output(2*P-1)) & multiplier_output;
```

Figure 4: Multiplier output extension

Counter

The Counter.vhd module consists of a ripple_carry_adder and a d-flip flop. Its operation is based on the fact that at each clock the previous output of the adder is added with the new input fw and therefore in output there is k*fw with k being the number of sums made by the adder. In this way we obtain a sine wave controlled with the input fw frequency.

```
entity Counter is
  generic (
    N : natural := 8
  );
  port (
    clk       : in  std_logic;
    a_rst_h   : in  std_logic;
    increment : in  std_logic_vector(N - 1 downto 0);
    en        : in  std_logic;
    cntr_out  : out std_logic_vector(N - 1 downto 0)
  );

end entity;
```

Figure 5: Counter.vhd module

```
FULL_ADDER_N_MAP : ripple_carry_adder
  generic map (Nbit => N)
  port map (
    a    => increment,
    b    => q_h,
    cin  => '0',
    s    => fullAdder_out,
    cout => open
  );

DFF_N_MAP : DFF_N
  generic map (N => N)
  port map (
    clk     => clk,
    a_rst_h => a_rst_h,
    d       => fullAdder_out,
    en      => en,
    q       => q_h
  );

-- Connect the output
cntr_out <= q_h;
```

Figure 6: Counter module made of an adder and a dff

Phase Adder

The Phase_Adder.vhd module takes care of obtaining input from the k*fw counter and adding the input phase to it to obtain the index as output to perform the lookup within the lookup table.

```
entity Phase_Adder is
  generic (N : integer := 16); -- Dimensione del segnale in bit
    Port (
      clk     : in  std_logic;
      a_rst_h : in  std_logic;

      signal_in : in  std_logic_vector (N-1 downto 0); -- Input segnale a N bit (std_logic_vector)
      phase_in  : in  std_logic_vector (N-1 downto 0); -- Input fase a N bit (std_logic_vector)
      signal_out : out std_logic_vector (N-1 downto 0) -- Output segnale modulato (std_logic_vector)
    );
end Phase_Adder;
```

Figure 7: Phase_Adder module

The module is composed of a ripple_carry_adder which takes care of adding the two inputs and a register for the output.

```
  FULL_ADDER_N_MAP : ripple_carry_adder
  generic map (Nbit => N)
  port map (
    a    => signal_in,
    b    => phase_in,
    cin  => '0',
    s    => adder_out,
    cout => open
  );

  PHASE_OUTPUT_REG: process(clk, a_rst_h, adder_out, output_reg)
  begin
    if (a_rst_h = '1') then
      output_reg <= (others => '0');
    elsif (rising_edge(clk)) then
      output_reg <= adder_out;
    end if;
  end process;

-- Connect the output
signal_out <= output_reg;
```

Figure 8: Phase adder module composed of an adder

lut_table_65536_7bit

The lut_table_65536_7bit.vhd module consists of a precompiled table with 16-bit input and 7-bit output. The module simply takes care of doing a lookup inside the table and returning the output associated with the input index.

```
entity lut_table_65536_7bit is
  generic (
    N : integer := 16;
    P : integer := 7
  );
  port (
    address  : in  std_logic_vector(N-1 downto 0);
    lut_out : out std_logic_vector(P-1 downto 0)
  );
end entity;
```

Figure 9: lut_table_65536_7bit module

Amplitude Multiplier

The Amplitude_Multiplier.vhd module takes care of multiplying the output signal from the LUT by the input amplitude in order to obtain the amplitude modulated result.

```
entity Amplitude_Multiplier is
  generic (
    N : natural := 7
  );
  port (
    clk     : in  std_logic;
    a_rst_h : in  std_logic;

    a : in  std_logic_vector(N - 1 downto 0);
    b : in  std_logic_vector(N - 1 downto 0);
    mul_out  : out std_logic_vector(2*N - 1 downto 0)
  );

end entity;
```

Figure 10: Amplitude multiplier module

The module is composed of a multiplier that multiplies the signal and the amplitude,

both on 7 bits, obtaining the result on 14 bits. The width is first extended from the original 4 bits to 7 bits. After the multiplier there is a register that manages the output.

```
MULTIPLIER_N : multiplier_NxN
  generic map (N=> N)
  port map (
      A => a,
      B => b,
      P => mul_nxn_out
  );

  AMPLITUDE_OUTPUT_REG: process(clk, a_rst_h, mul_nxn_out, output_reg)
  begin
    if (a_rst_h = '1') then
      output_reg <= (others => '0');
    elsif (rising_edge(clk)) then
      output_reg <= mul_nxn_out;
    end if;
  end process;

-- Connect the output
mul_out <= output_reg;
```

Figure 11: Amplitude multiplier composed of a multiplier_NxN

QLUT Optimization

To perform an optimization regarding the space occupied by the LUT, which with 16 input bits begins to have considerable dimensions, the QLUT Optimization technique was adopted which consists in using only the first quadrant of the lookup table, thus saving a large amount of memory, denying the input and/or output of the LUT depending on the value of the last two bits of the input signal.

```
lut_address <= signal_out(N-3 downto 0) when (signal_out(N-2) = '0') else not(signal_out(N-3 downto 0));

LUT_16384 : qlut_table_16384_7bit
  generic map (N => N-2, P => P)
  port map(
    address  => lut_address,
    lut_out => lut_output
  );

  lut_output_mux <= lut_output when (signal_out(N-1) = '0') else not(lut_output);
```

Figure 12: QLUT Optimization

This technique allowed, with the same results, to have a considerable saving on occupied memory and lookup time.

4. Test Phase

To check the correct functioning of the multi-standard modulator, a testbench was developed to display the signal generated as the frequency, amplitude and phase vary and to check the correct functioning of the reset signal.

The testing phase was divided into various phases, each of which varied an input to then study the wave as it varied and ensure that the modulation occurred correctly.

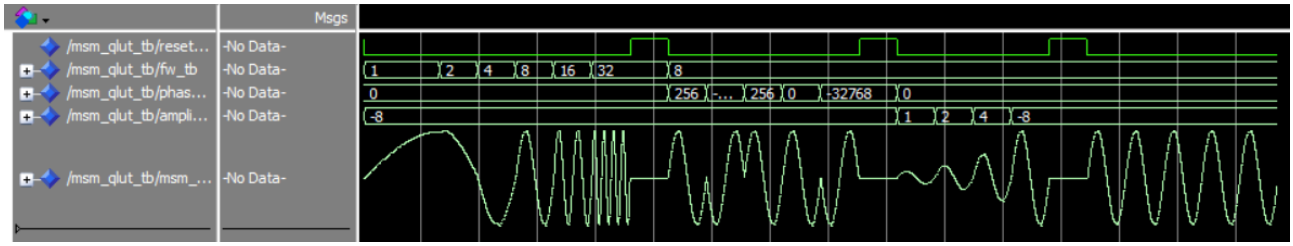

Figure 13: Test phase

Frequecy modulation

In this first phase the frequency is varied. It is clearly seen that as the input frequency varies, the output wave also varies and is therefore modulated in frequency.

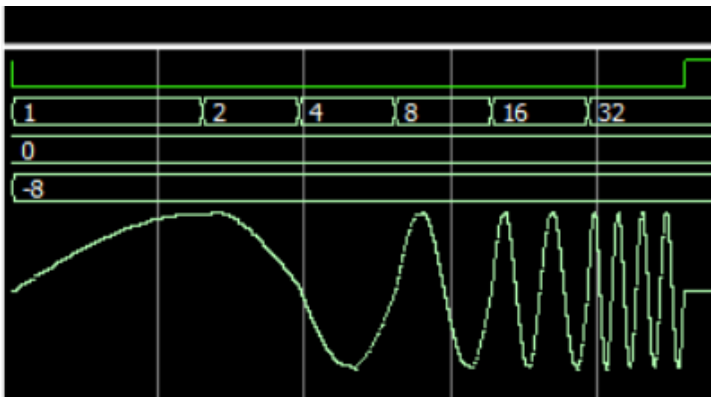

Figure 14: Frequency varying

Phase modulation

In this phase the phase of the wave is varied based on the input phase. It can be clearly seen that as the input varies, the phase of the outgoing wave also varies, which is therefore modulated in phase.

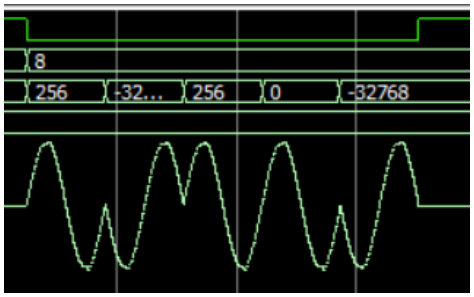

Figure 15: Phase varying

Amplitude modulation

In this phase the amplitude input to the system is varied. It is clearly seen that as the input amplitude varies, the output wave also varies, which is therefore modulated in amplitude.

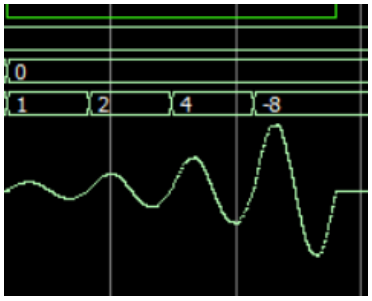

Figure 16: Amplitude varying

Reset phase

Tests were also done for the reset signal. As you can see, when the reset signal is active the system output is zero, therefore it can be stated that the reset is working correctly.

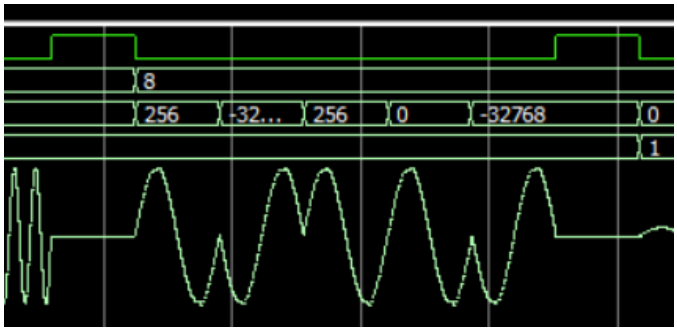

Figure 17: Reset working

QLUT Comparison

To prove that the results obtained with the LUT and the QLUT are identical, both waves were printed simultaneously. We can clearly see that the wave resulting from both lookup tables are identical and therefore we managed to obtain the same result with a lookup table that is a quarter of the original one, thus saving memory and obtaining much lower lookup times.

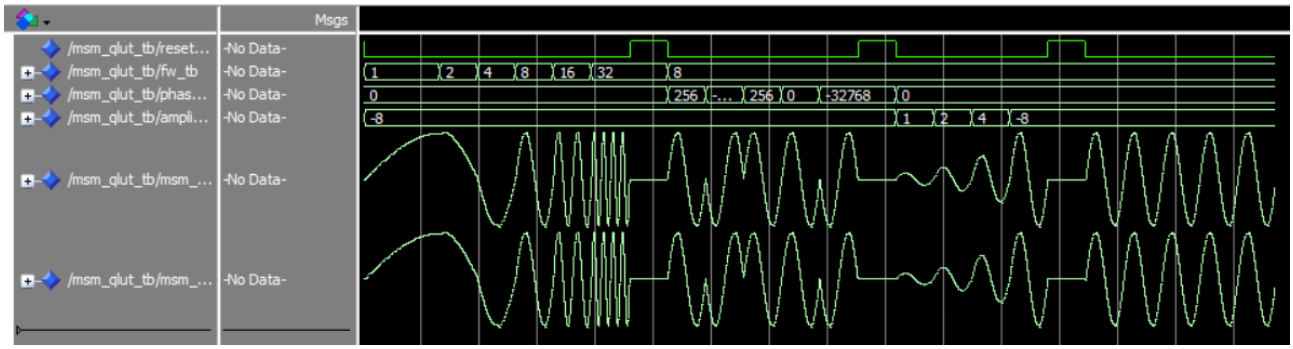

Figure 18: QLUT Comparison

5. Vivado report

Finally, the Vivado tool was used, in order to realize and analyze the following phases:

- Elaborated design analysis
- Implementation analysis

As working device, the xc7z010clg400-1 FPGA was selected.

Elaborated design

The following figure shows the Register Transfer Level description of the architecture:

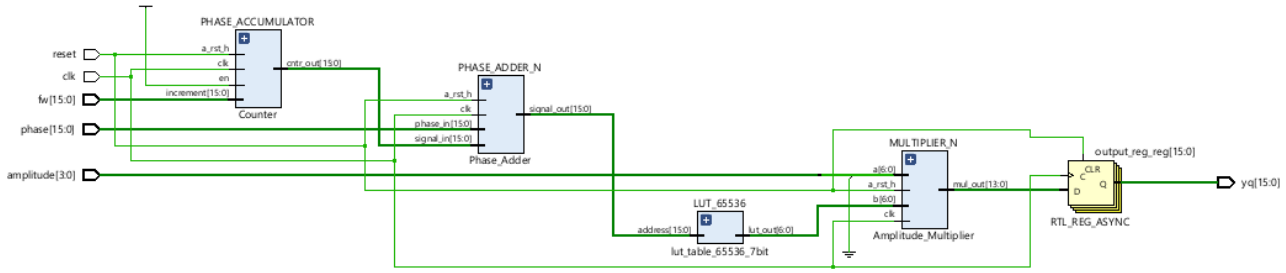

Figure 19: RTL Architecture

Implementation

During the synthesis and the implementation phase, the modules described with VHDL are mapped to circuits that the FPGA is able to implement, exploiting the components in the Netlist.
Vivado tries to do that respecting the given constraints. In this case the only constraint we had was the clock period, that it is chosen to have a period of 10 *ns*, so a frequency of 100 MHz.

Timing and Critical path

The implementation timing report is the following:

Design Timing Summary

Setup		Hold		Pulse Width	
Worst Negative Slack (WNS):	0,239 ns	Worst Hold Slack (WHS):	0,198 ns	Worst Pulse Width Slack (WPWS):	4,500 ns
Total Negative Slack (TNS):	0,000 ns	Total Hold Slack (THS):	0,000 ns	Total Pulse Width Negative Slack (TPWS):	0,000 ns
Number of Failing Endpoints:	0	Number of Failing Endpoints:	0	Number of Failing Endpoints:	0
Total Number of Endpoints:	63	Total Number of Endpoints:	63	Total Number of Endpoints:	64

All user specified timing constraints are met.

Figure 20: Timing report

From the timing report we can see that all the constraints relating to the clock have been respected and this means that the right clock has been chosen that respects the timings of the various components of the system.

Utilization

The utilization report is the following:

Name	Slice LUTs (17600)	Slice Registers (35200)	F7 Muxes (8800)	F8 Muxes (4400)	Slice (4400)	LUT as Logic (17600)	Bonded IOB (100)	BUFGCTRL (32)
N MSM	418	63	55	13	122	418	54	1
PHASE_ADDER_N (Phase_Adder)	148	19	0	0	89	148	0	0
PHASE_ACCUMULATOR (Counter)	29	16	0	0	14	29	0	0
DFF_N_MAP (DFF_N)	29	16	0	0	14	29	0	0
MULTIPLIER_N (Amplitude_Multiplier)	19	12	0	0	15	19	0	0
MULTIPLIER_N (multiplier_NxN)	19	0	0	0	15	19	0	0
LUT_65536 (lut_table_65536_7bit)	223	0	55	13	94	223	0	0

Figure 21: Utilization report

From the utilization report we can see the resources used by the various components within the system, such as the LUTs used, registers and so on.

Power

The power analysis summary is the following:

Summary

Power analysis from Implemented netlist. Activity derived from constraints files, simulation files or vectorless analysis.

Total On-Chip Power:	**0.122 W**
Design Power Budget:	**Not Specified**
Power Budget Margin:	**N/A**
Junction Temperature:	**26,4°C**
Thermal Margin:	58,6°C (5,0 W)
Effective ϑJA:	11,5°C/W
Power supplied to off-chip devices:	0 W
Confidence level:	Low

Launch Power Constraint Advisor to find and fix invalid switching activity

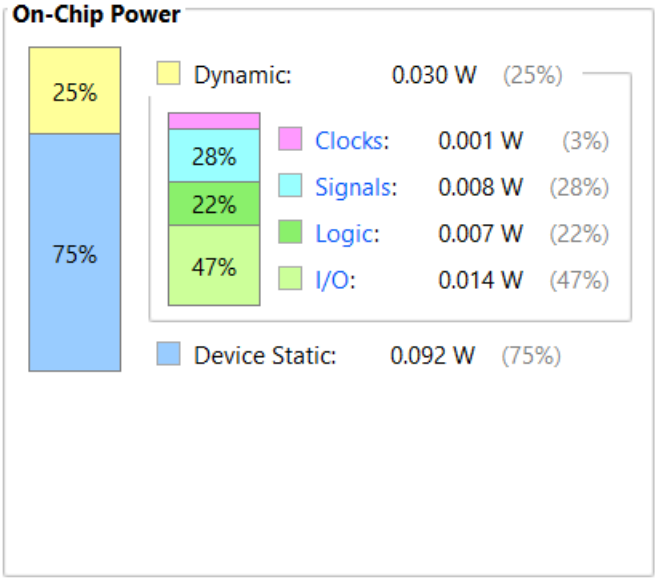

Figure 22: Power analysis report

From the power analysis summary, we can see the energy consumption of the various components of the system.

Messages and warnings

The messages and the warnings shown by Vivado are the following:

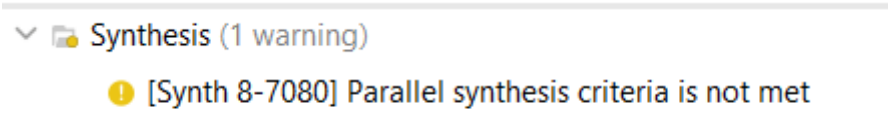

Figure 23: Message warning

In Vivado, the warning message "[Synth 8-7080] Parallel synthesis criteria is not met" indicates that the project does not meet the criteria necessary to take advantage of parallel synthesis. Parallel synthesis is a feature that allows Vivado to split synthesis work across multiple processes, potentially speeding up total synthesis time.
Therefore, it is a warning message that is not necessarily linked to the system, but more to the Vivado synthesis process.

6. Conclusion

In this project, a Multi standard Modulator has been correctly developed, i.e. a modulator capable of performing FSK, PSK and ASK modulation. These types of modulation can be performed by simply varying the frequency, phase and/or amplitude at the input to the system so as to obtain the correctly modulated signal at the output.
The system is implemented using a LookUp Table (LUT), which however can be a bottleneck given that, with 16-bit input, the LUT becomes of the order of 2^16, causing memory occupation and a lookup time not negligible. Therefore, the QLUT Optimization technique was used to have a LUT of the order of 2^12, i.e. a quarter of the size of the LUT without changing the result of the original LUT. This approach allowed us to obtain better performance in terms of occupied space and lookup time without changing the final result.
The result obtained from this process is therefore a system capable of modulating in amplitude, phase and/or frequency which finds application in an immense number of fields, as we have seen in the possible applications, making the system a reliable and usable device to be used during many types of communications.