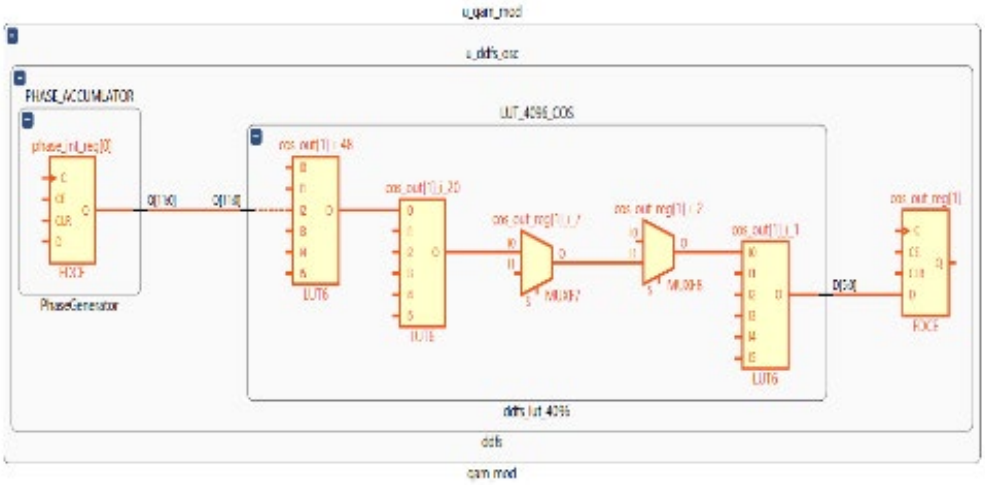
M'ary Quadrature Amplitude Modulator IP in VHDL, FPGA and SDR

Fully verified and synthesized with source code and netlist, integration guide, license and manual



Angelia Technologies

Mark Chen

Feb. 12, 2023

Introduction

The modulator is designed using VHDL to realize the M-QAM modulation.

This document presents an implementing for Quadrature Amplitude Modulation (QAM) signal modulators using Field Programmable Gate Array (FPGA).

Purpose of the project is to investigate the low-level "RTL based" implementation and the building blocks of the QAM modulator. Therefore, to provide a flexible way of testing the transmission and modulation properties later in real-time, a higher level hierarchical implementation based on GNURadio was provided to investigate future application of offloading QAM modulation into FPGA, to accelerate the development, testing, and verification of SDR (Software-Defined Radio) application in real-time environment.

Each system module will be presented, and verified with testbenches along with Matlab scripts to verify the functional behavior of the QAM waveform including the constellation diagram and the upconverted passband waveform transition, then with respect to all system components. In this system no target board for implementation has been specified, but it addresses only to the Zynq-7000 FPGA.

Design Implementation Flow

Implementation of the entire systems was done in VHDL without the help of IP-Core Package of Xilinx System Generator or DSP Builder Tools.

Design flow comprises of the following steps:

Architecture design,
VHDL design entry,
Behavioral simulation,
Synthesis,
Implementation,
Timing analysis and
Download to ZYBO board*

Bitstream for the board will not be generated, and the synthesis has been performed without the IO planning.

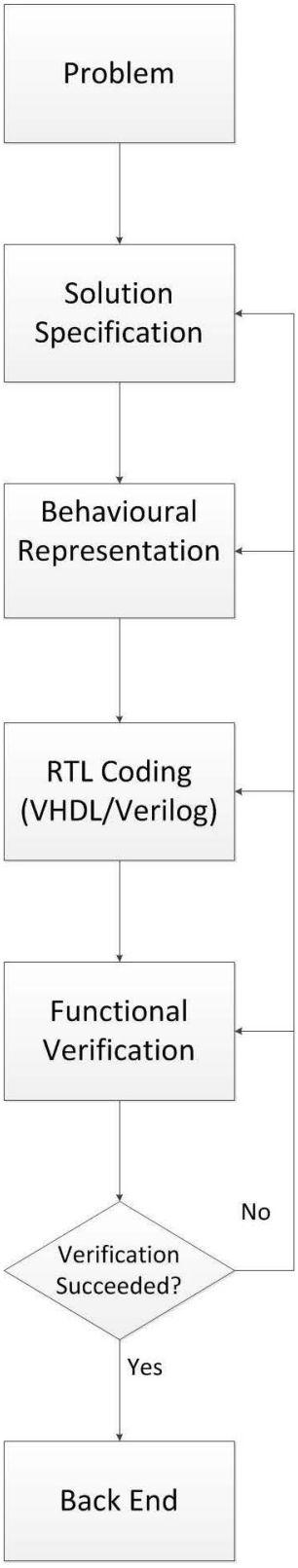


Figura 1- Frontend flow

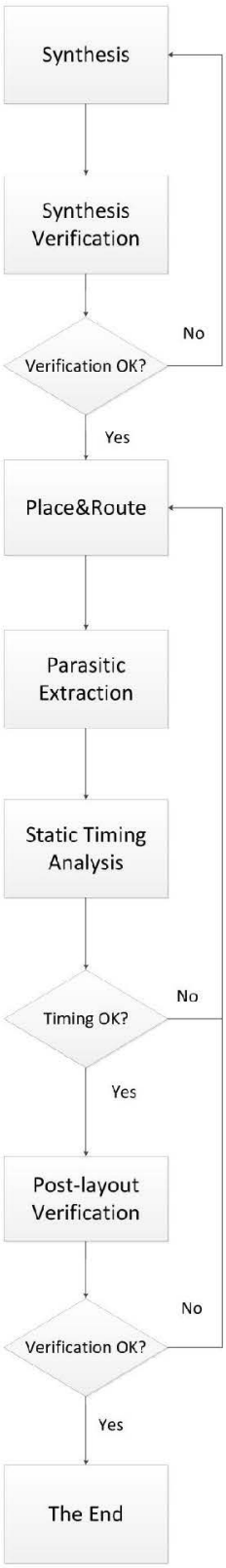


Figura 2 - Backend flow

Design, Simulation & Synthesis Software

Xilinx Vivado Webpack

ModelSim and GHDL simulator

GTViewer and Scansion waveform viewer.

Design Specifications

QAM has different types of constellations.

Some of the constellations are called 4-QAM, 8-QAM, 16-QAM and 64-QAM .. etc based on the nearest point to origin. For example the 4-QAM constellation is limited to the 4 nearest points around the origin.

When the constellation is increased, the number of amplitude and phase are also increased. Its frequency utilization (expressed in bits per second per hertz) is also more efficient. In this project we focus on the implementation of 4-QAM, however, it' easy to extend the order by only replacing the mapper module.

The basic operation of modulator is based on signal mixing which means multiplying the input signal and a reference oscillator signal, to produce spectral images at the sum and difference frequencies. The implemented QAM modulator addressed here is a 4-QAM based modulator where each symbol is represented by 2 bits, with respect to the following formula, we got 2 bits per each symbol, where M is the modulation order which is equal to 4.

Bits_Per_Symbol = $\log_2(M)$

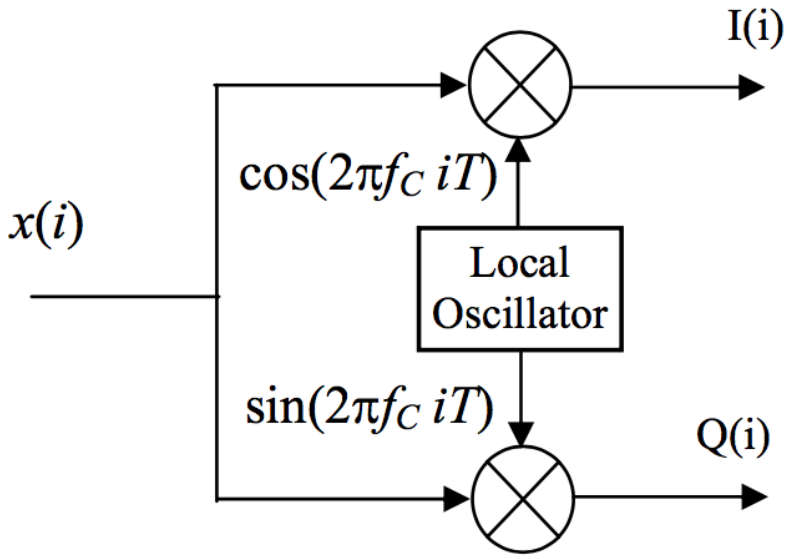


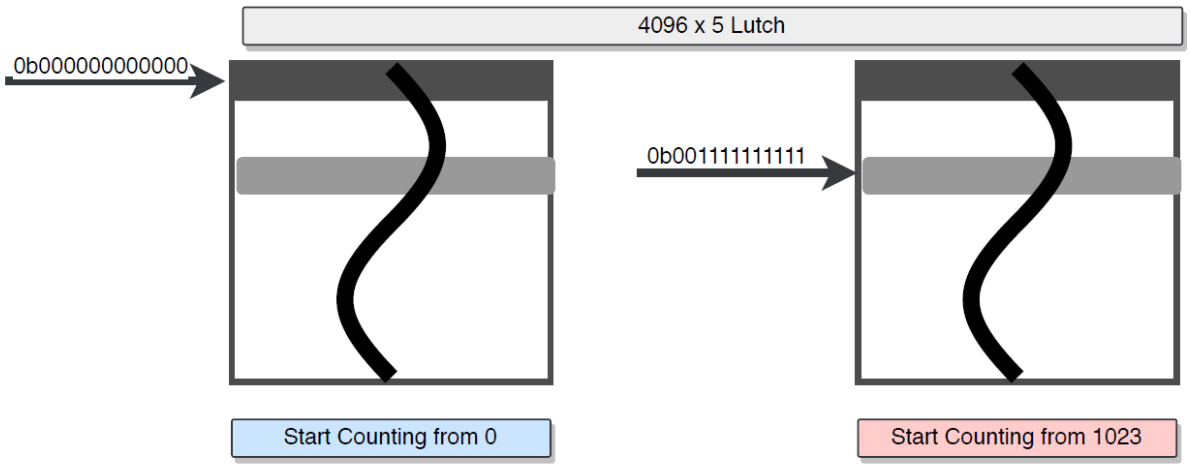
Figure 3. QAM Modulator Block Diagram

One of the main tasks in implementing any digital transmitter including QPSK, 4-QAM, 16-QAM .. etc is the generation of the sine wave carrier. In order to do that, a 12-bit phase accumulator and Look Up Table (LUT) has been used based on Direct Digital Frequency Synthesizer (DDFS). After getting the carrier generated, the basic

QAM modulated signal is nothing more than the carrier signal itself at different phases and constant amplitude.

For the implementation of 4-QAM modulators, two sinusoidal carriers are needed - a sine wave and cos wave as described in Fig 3.

To get the two carriers with 90-degree phase shifts, only 1 phase accumulator was used but with different addressing ways to fetch the samples for both the sine. and cosine as illustrated in Fig 19.



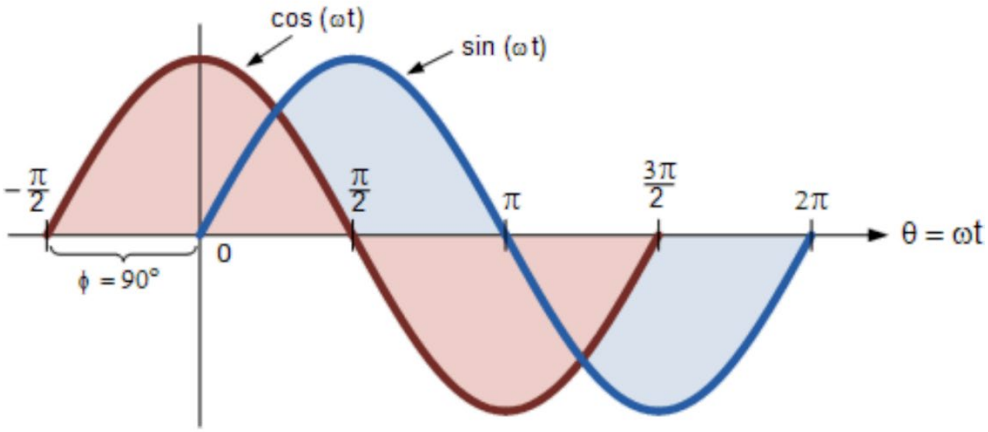


Figure 4.Waveform Oscillator

For I balanced modulator, 2 phases are possible:
+sin(ωct) and -sin(ωct)

For Q balanced modulator, 2 phases are possible:
+cos(ωct) and -cos(ωct)

Linear summer will combine the 2 quadrature signals and yet 4 possible phases act as an output as shown in Fig 5

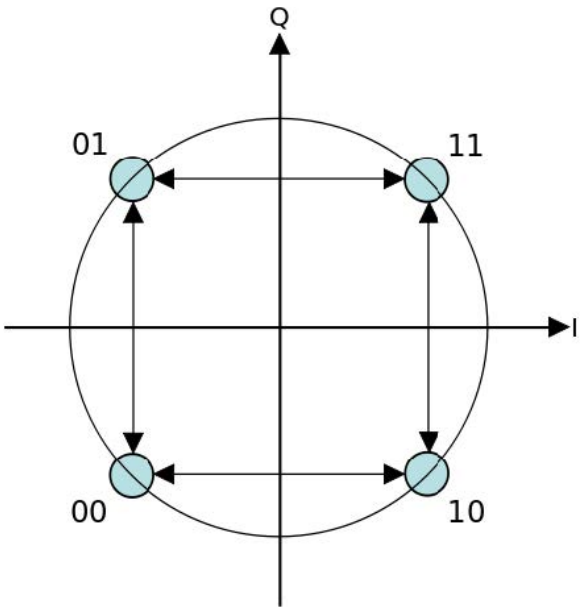


Figure 5. 4-QAM Constellation

Digital Oscillator

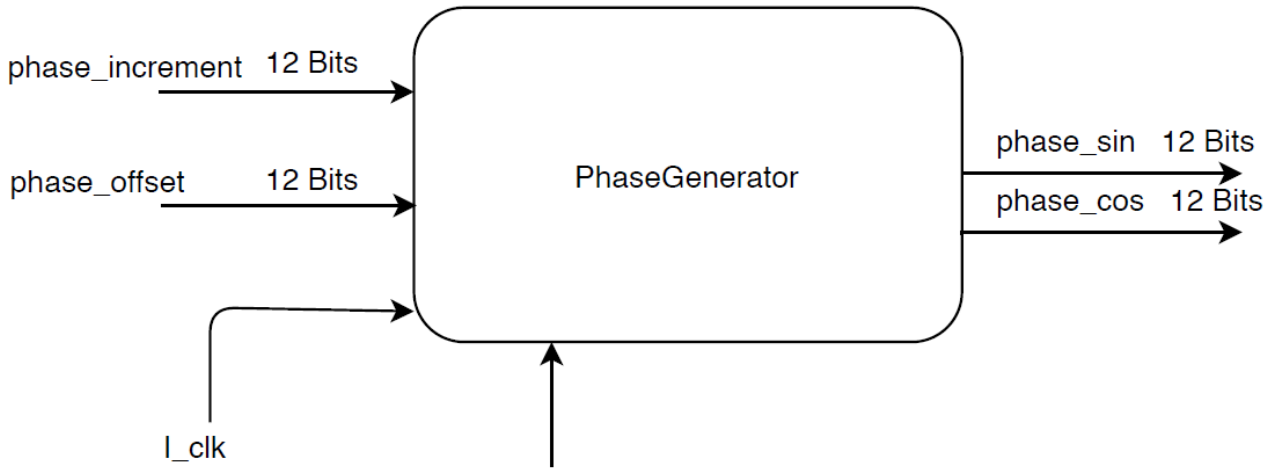


Figure 6. Digital Oscillator

This module is basically a phase accumulator which is responsible for adding a constant value FCW (Frequency Control Word) which is indicated here under the name phase_increment.

The oscillator wraps around every time it reaches its maximum value. For example, in our configuration where the maximum value is 4096 (12-bit accumulator) and the phase_increment is 4, the accumulator will count: 0, 1, 2, 3, 4, ..., 4095. The system in this configuration counts from 0 to address later the sinusoidal waveform from a latch. As the address counter steps through each memory location, the corresponding digital amplitude of the signal at each location. On the other hand, it also starts counting from 1023 to generate the cosine wave as illustrated in Fig 19.

4-QAM Constellation Mapper

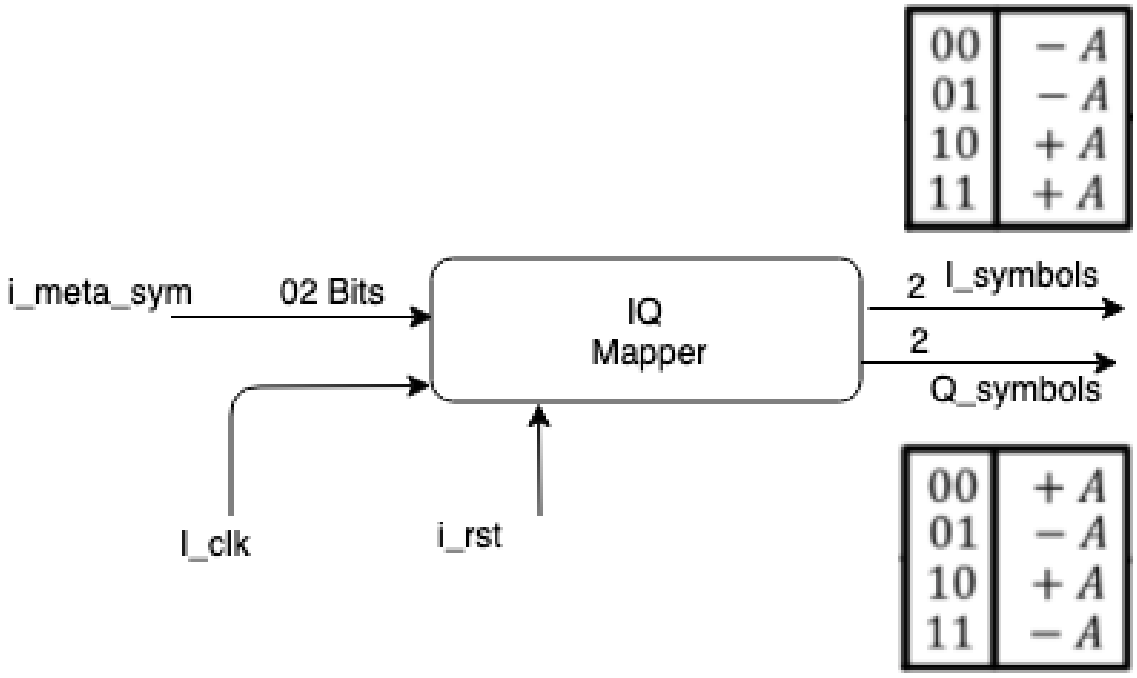


Figure 7. 4-QAM Constellation Mapper

This module is responsible for constellation mapping of 4-QAM symbols. We want to have one single change of bits from one symbol to the other; thus, using fixed point number[gray code] representation 00, 01, 11, 10 to represent 1+1j, -1+1j, -1-1j, 1-1j as shown in Fig 7, see Table 1.

Table 1. 4-QAM Symbol Mapping

Constellation[Binary Input Sequence]	Symbols Mapping
00	**+1+J**
01	**-1+J**
10	**+1-J**
11	**-1-J**

QAM Modulator

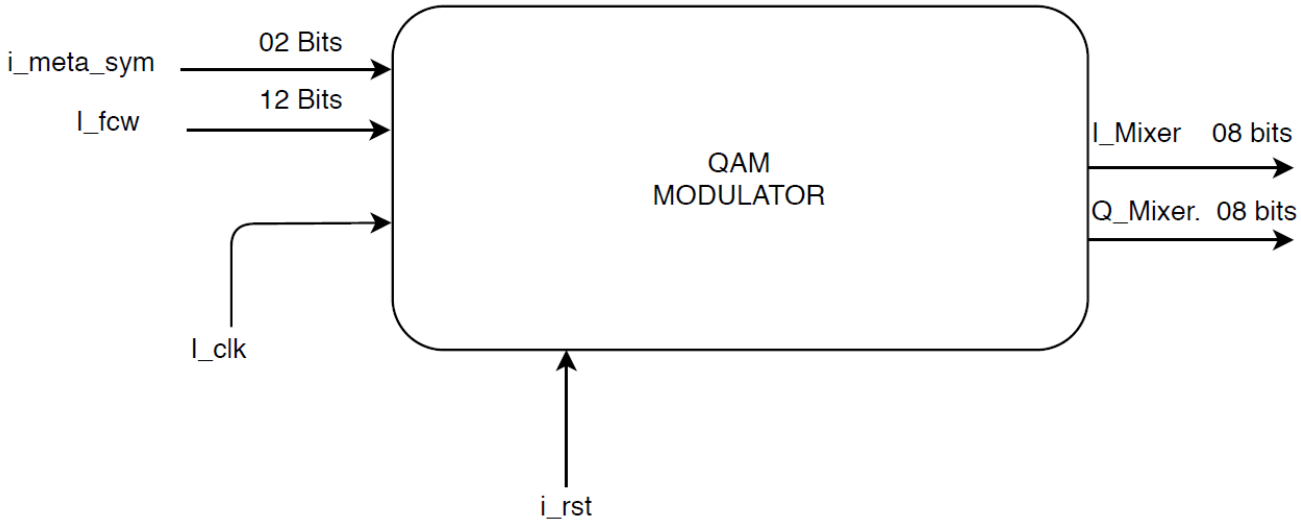


Figure 8. QAM Modulator Module

The QAM Modulator module takes as an input the a sequence pair of bits and the frequency control word and then performs the multiplication or in other words, the mixing process to produce the passband upconverted signals for both I component and Q components.

For I balanced modulator, 2 phases are possible:

+sin(ωct) and - sin(ωct)

For Q balanced modulator, 2 phases are possible:
+cos(ωct) and - cos(ωct)

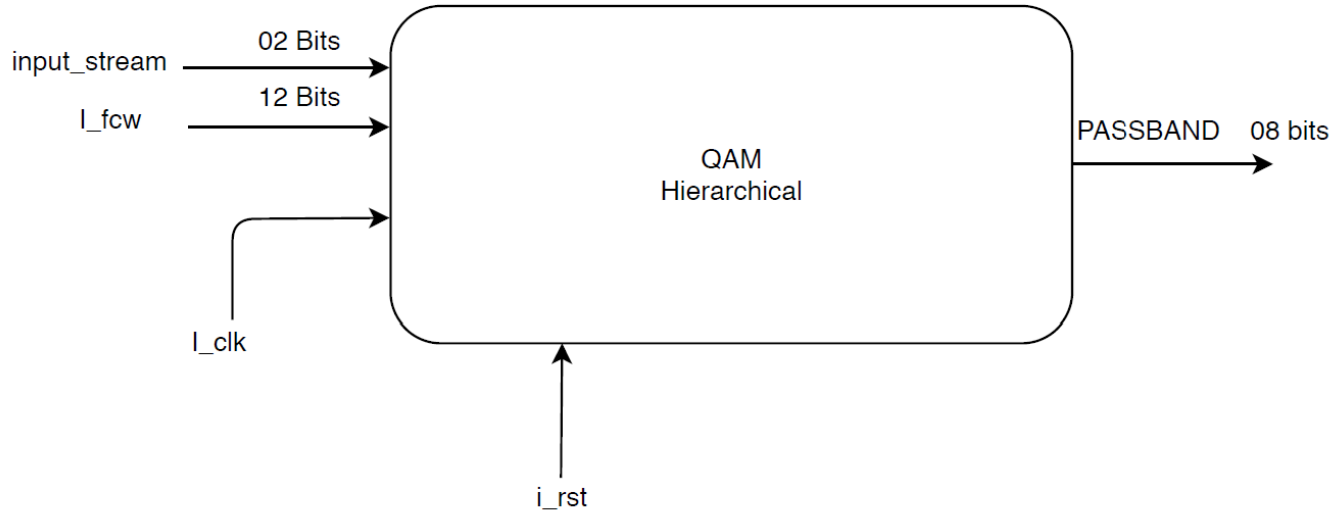


Figure 9. QAM Encapsulator

This module is nothing than an encapsulator for the QAM modulator module. It performs linear summation addition by combining the up-converted signals.

Project Directory Structure

Files
Documentation files

Verifications
Simulation and Verification scripts

Matlab
QAM constellation and modulator behavior verification

SDR
Constellation verification in realtime using USRP.

ModelSim_ghdl_src_DUT
Files to be tested using ModelSim, ghdl (for test run ./compile) together with VHDL source codes.

QAM_Vivado
Vivado Project (timing constraints, source codes also included within the project)

VHDLSrc
VHDL Source files

Testbenches
VHDL testbench files

Synthesis
Synthesis results with VCD files to view waveforms

README
User Manual
Integration Guides
License

Key VHDL Source Files

qam_mod_tb.vhd

QAM modulator testbench

QAM_Hierarchical.vhd

This Module Is responsible for adding the two generated passband waveforms:
I_Symbol mixedwith coswave + Q_Symbol mixedwith sinwave

qam_mod.vhd

The system takes the input I and Q symbols each 2 bits and produces the modulated waveforms for each of the I and Q components .

qam_mapper.vhd

QAM mapper: constellation mapper of 4-QAM
We want to have one single change of bits from one symbol to the other; thus, using fixed point number[gray code] representation {00, 01, 11, 10} to represent {1+1j, -1+1j, -1-1j, 1-1j}

PhaseGenerator.vhd

Phase generator/phase accumulator
The system is responsible to start counting once the system is started; with different offset e.g counting from 0 to generate the sine wave (index the latch).
Counting from 1024 (equivalent to pi/2) to generate a cos wave.

full_adder.vhd

FullAdder: 1-Bit Ripple Carry Adder Module

FA_N.vhd

FullAdder: N- Ripple Carry Adder Module

IQ_Adder.vhd

This module is responsible for adding the two generated passband waveforms:
I_Symbol mixedwith cos wave + Q_Symbol mixedwith sin wave

ddfs.vhd

DDFS: Direct Digital Frequency Synthesizer

ddfs_lut_4096.vhd

Sinusoidal latch 4096x12, generated using Matlab

RTL Verification

The behavioral simulation have been achieved using GNU-Radio and Matlab along with testbenches developed using ghdl and scansion to fully cover all of the test corner cases in each module (which were actually bounded with respect to the number of test-vectors).

Simulation Scripts

Bash script compile.sh, VHDL verification (simulation), compilation file using ghdl compiler under OS X.

QAM_Vivado – Simulation Project

SDR gnuradio Implementation

qam_mod.py

GNU Radio Python Flow Graph
QAM Modulator Constellation Verification

Matlab Verification

qam_verification.m

Matlab code to verify the behavior of 4-QAM modulation scheme.

Waveform

Figure 10 are illustrated the waveforms of the Phase Accumulator Counter where each counter is associated with a waveform, i.e. for cosine wave generation we count from 1023 which is equivalent to pi/2 to later index the latch and get the cosine data points as shown. On the other hand, to generate a sine wave we start the default counting from 0 and therefore we fetch the sine wave data points.

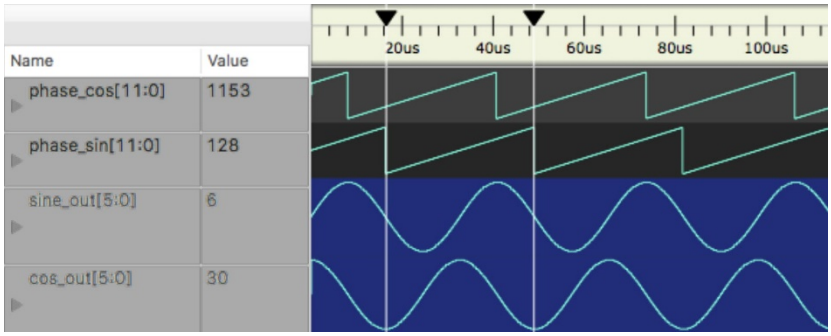


Figure 10. Digital Oscillators: DDFS

Figure 11 illustrates the individual modulated waveforms for I and Q component after performing the mixing process. Hence: since we mix (multiply) a waveform of size **N** and a symbol containing 2 bits, size **M** we got an output vector of size **M+N**.

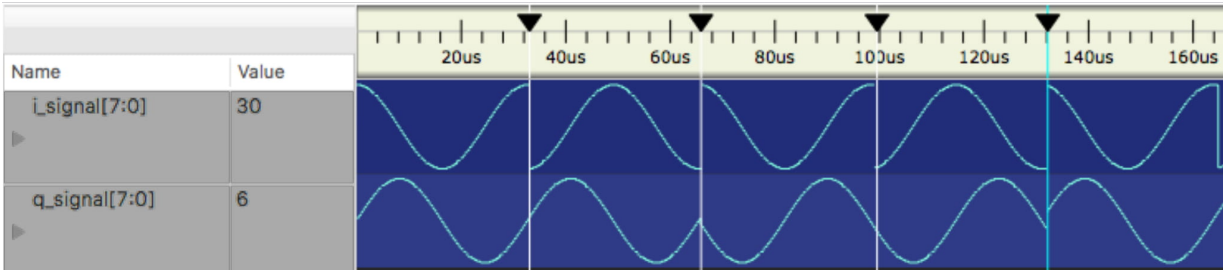
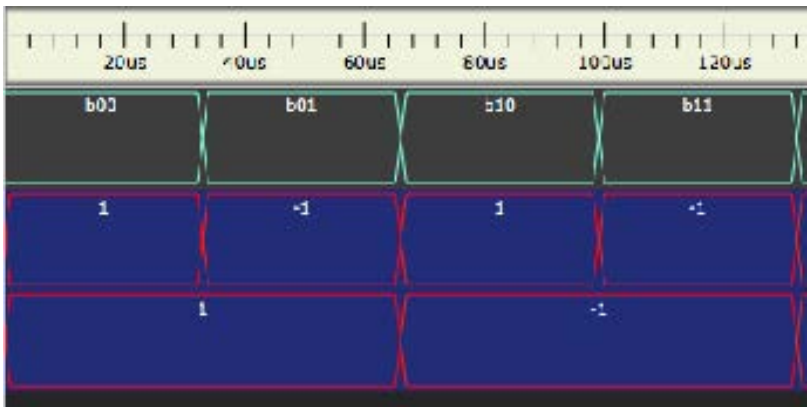


Figure 11. Passband: Upconverted Waveforms

In figure 12 it is shown the constellation behavior with respect to the input symbols as described before in Table 1 versus the Matlab equivalent verification.



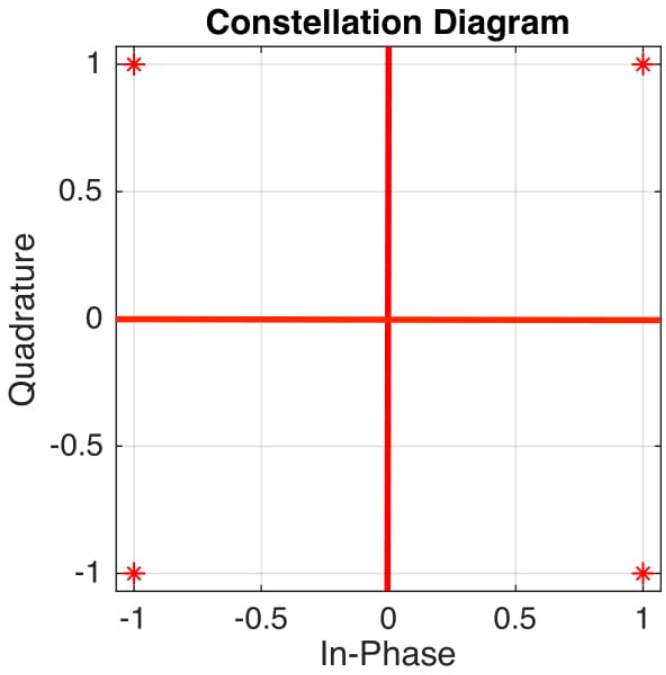
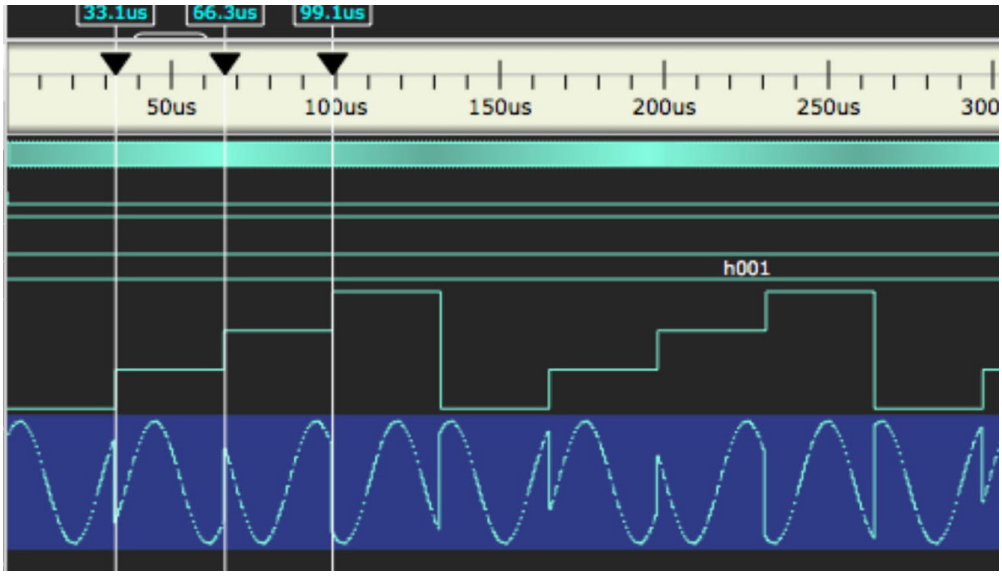


Figure 12. [VHDL vs MATLAB] 4-QAM Constellation Diagram

In Figure 13 it is shown the final stage of behavioral verification using the testbench and Matlab for the waveform transition with respect to the input symbols.



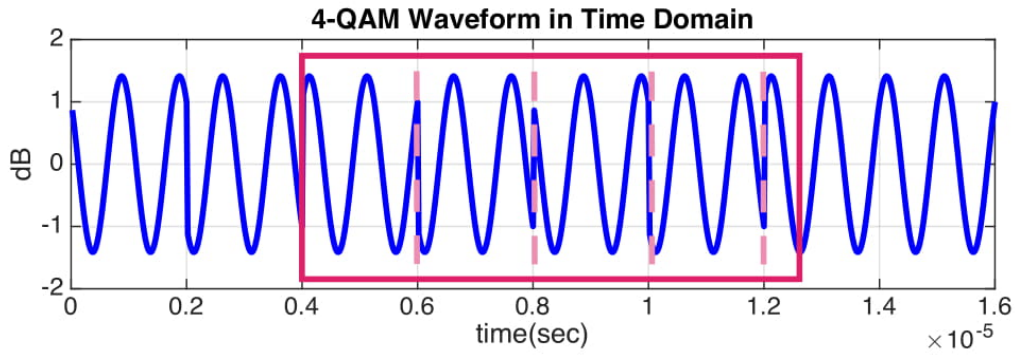


Figure 13. [VHDL vs MATLAB] 4-QAM Modulated Signal

Synthesis

Synthesis and Implementation have been achieved using Xilinx Vivado.

The synthesis has been performed versus different clock based criterion. The main selection criterium for the clock constrains is the assumption that we want the system to operate in 125MHz.

However different tests have been done to try to achieve a maximum clock frequency of 200MHz and 614.4 MHz.

The schematic of the system after elaborating the design is shown in figures 14, 15 and 16. which are basically reading RTL files (VHDL files) and recognizing code syntax that represent real hardware structures. Once recognized, these are converted (in Vivado synthesis case) into generic technology cells - abstract things like registers, adders, compactors, multiplexers, gates, etc...

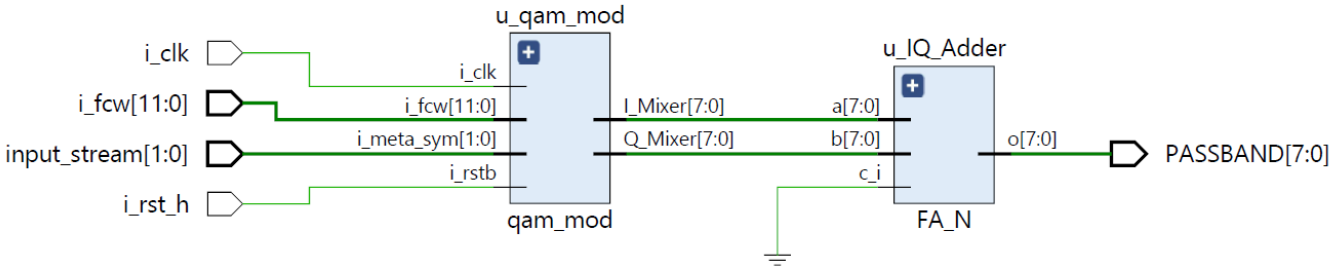


Figure 14. QAM Higher Level Schematic

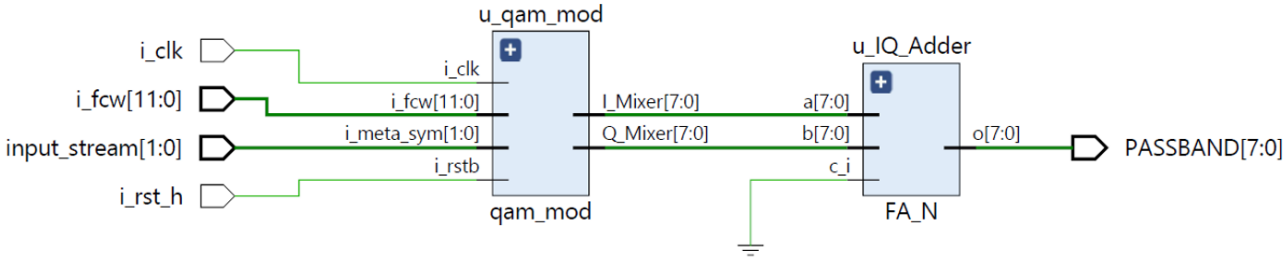


Figure 15. qam_mod Block Schematic

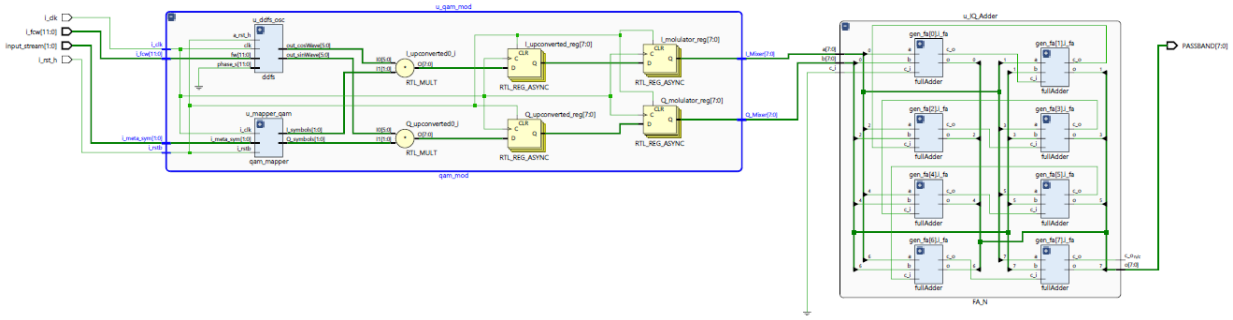


Figure 16. IQ_Adder Block Schematic

After elaborating the design, the synthesis is performed with no constraints first to verify that every thing is working correctly and that the design is bug-free. Then we defined the constraints by defining the clock period to be **8 nano sec** to achieve a maximum clock frequency of 125MHz. Then the implementation is performed, where implementation stage is intended to translate netlist into the placed and routed FPGA design. Xilinx design flow has three implementation stages: translate, map and place and route. (These steps are specific for Xilinx.

Figure 17 reportes the utilized resources. Hint: Extended timing summary is listed in Annex, **Table 2. Timing_Report_Summary**

Resource	Utilization	Available	Utilization %
LUT	285	17600	1.62
FF	83	35200	0.24
IO	24	100	24.00

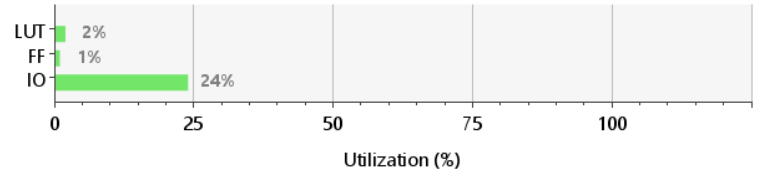


Figure 17. Utilization Report Summary

Below in Fig. 18 it is reported the critical path due to the minimum period which in turns indicates the maximum operating frequency. Also shown all possible paths in **Figure 19.**

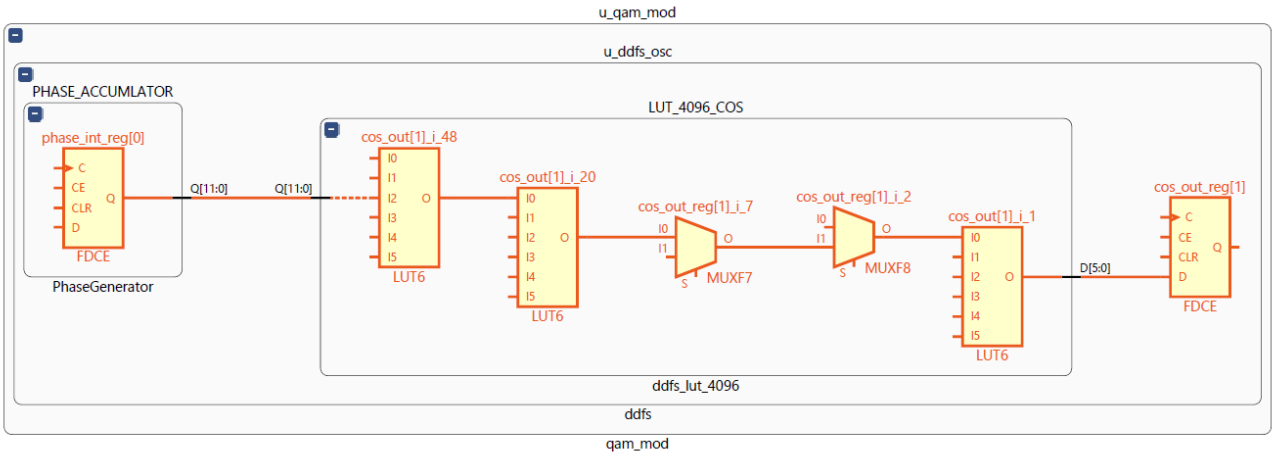


Figure 18. Critical Path Schematic

Name	Slack	Levels	Routes	High Fanout	From	To	Total Delay	Logic Delay	Net Delay	Requirement
Path 1	2.386	5	3	52	u_qam_mod/u...nt_reg[0]/C	u_qam_mod/u...t_reg[1]/D	5.516	1.519	3.997	8.0
Path 2	2.603	5	4	78	u_qam_mod/...g_rep[2]/C	u_qam_mod/u...ut_reg[3]/D	5.323	1.565	3.758	8.0
Path 3	2.691	5	3	78	u_qam_mod/u...nt_reg[2]/C	u_qam_mod/u...t_reg[0]/D	5.193	1.727	3.466	8.0
Path 4	2.717	5	4	78	u_qam_mod/...g_rep[2]/C	u_qam_mod/u...ut_reg[2]/D	5.207	1.565	3.642	8.0
Path 5	2.734	5	4	82	u_qam_mod/u...nt_reg[3]/C	u_qam_mod/u...t_reg[3]/D	5.114	1.396	3.718	8.0
Path 6	2.770	4	3	78	u_qam_mod/u...nt_reg[2]/C	u_qam_mod/u...t_reg[4]/D	5.189	1.335	3.854	8.0
Path 7	2.811	5	3	82	u_qam_mod/u...nt_reg[3]/C	u_qam_mod/u...t_reg[2]/D	5.089	1.612	3.477	8.0
Path 8	2.912	4	4	78	u_qam_mod/...g_rep[2]/C	u_qam_mod/u...ut_reg[1]/D	5.013	1.182	3.831	8.0
Path 9	3.076	5	3	52	u_qam_mod/...g_rep[0]/C	u_qam_mod/u...ut_reg[0]/D	4.881	1.427	3.454	8.0
Path 10	3.426	4	3	78	u_qam_mod/...g_rep[2]/C	u_qam_mod/u...ut_reg[4]/D	4.534	1.273	3.261	8.0

Figure 19. Paths

And lastly; in Fig. 20 we report the power consumption and the Worst Negative Slack by making a snapshot comparison between working on 125MHz Clock and 200MHz clock and clearly we can observe the increase of dynamic power consumption by increasing the operating system frequency.

P =1/2CV F

where P is the power, C is the load capacitance and V is the supply voltage level. The frequency refers to the clock frequency and data toggles once every clock cycle.

On the other hand, the Worst Negative Slack (WNS) reported in Fig. 20 is actually the worst positive slack. If WNS is positive then it means that the path passes. If it is negative, then it means the path fails. The "Total Negative Slack (TNS) is the sum of the (real) negative slack in the design. If it is 0, then the design meets timing. If it is a positive number, then it means that there is negative slack in the design (hence the design fails). It cannot be negative.

In both clock configuration the system has met the requirements constraints but further increasing the clock frequency to i.e 614.4 MHz will fail where different optimization techniques must be performed to meet the clock requirements like adding pipelines in the system design or the use of DSP Blocks on ZYNQ where it's observed from timing report in Fig 13 that no DSP Blocks have been used. Hence, the choice of clock 614.4 MHz was based on the waveform frequency resolution of 150KHz.

$\log_2(f_{clk}=DesiredFreq) = N$

where N is the number of points needed to represent the waveform, Desired Freq is 150KHz and fclk is the system clock constraints.

Design Timing Summary

Setup		Hold		Pulse Width	
Worst Negative Slack (WNS):	2,386 ns	Worst Hold Slack (WHS):	0,170 ns	Worst Pulse Width Slack (WPWS):	3,500 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:	80	Total Number of Endpoints:	80	Total Number of Endpoints:	84

All user specified timing constraints are met.

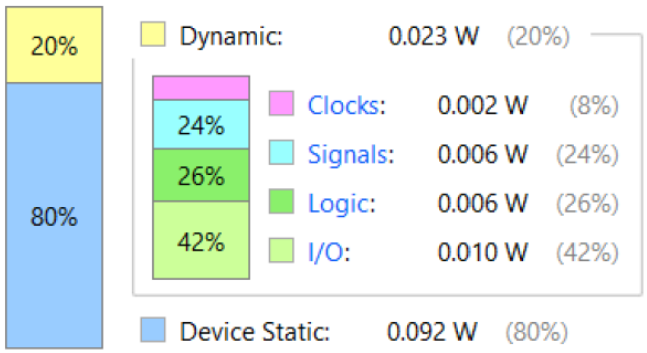
Summary

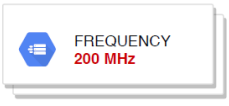
Power analysis from Implemented netlist. Activity derived from constraints files, simulation files or vectorless analysis.

Total On-Chip Power:	**0.115 W**
Design Power Budget:	**Not Specified**
Power Budget Margin:	**N/A**
Junction Temperature:	**26,3°C**
Thermal Margin:	58,7°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

On-Chip Power





Setup		Hold		Pulse Width	
Worst Negative Slack (WNS):	0,386 ns	Worst Hold Slack (WHS):	0,170 ns	Worst Pulse Width Slack (WPWS):	2,500 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:	80	Total Number of Endpoints:	80	Total Number of Endpoints:	84

All user specified timing constraints are met.

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

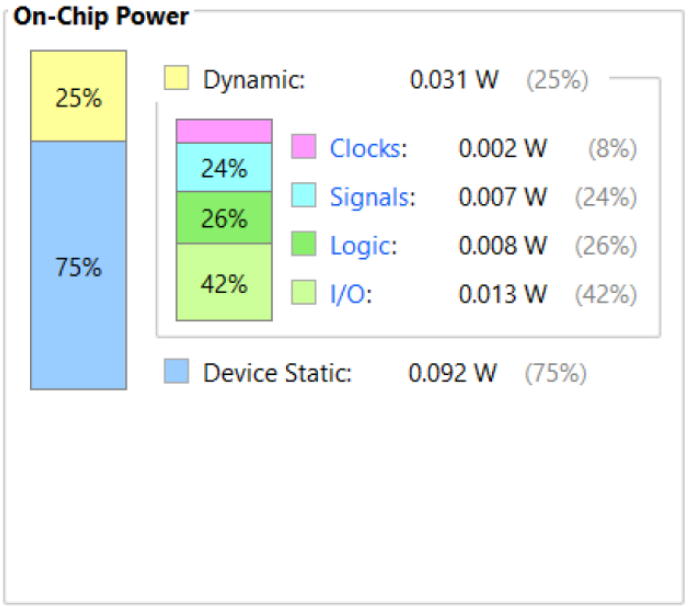


Figure 20 Power Consumption and the Worst Negative Slack

Finally, in Fig. 21 and Fig. 22 the device implementation is shown.

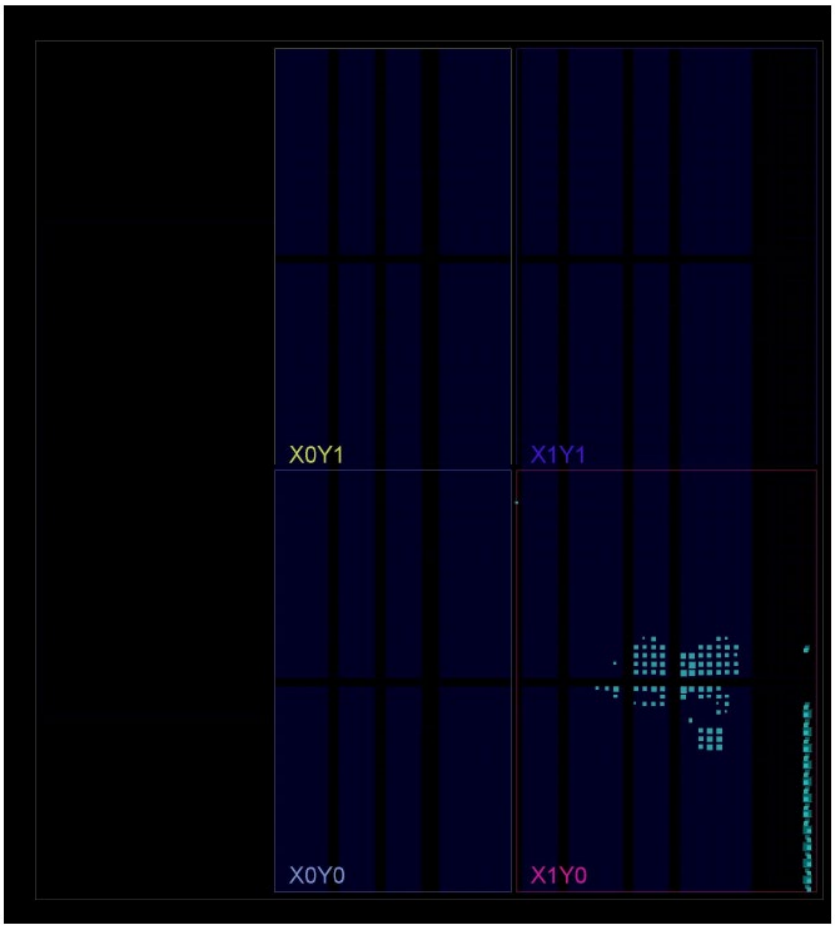


Figure 21. Device Implementation

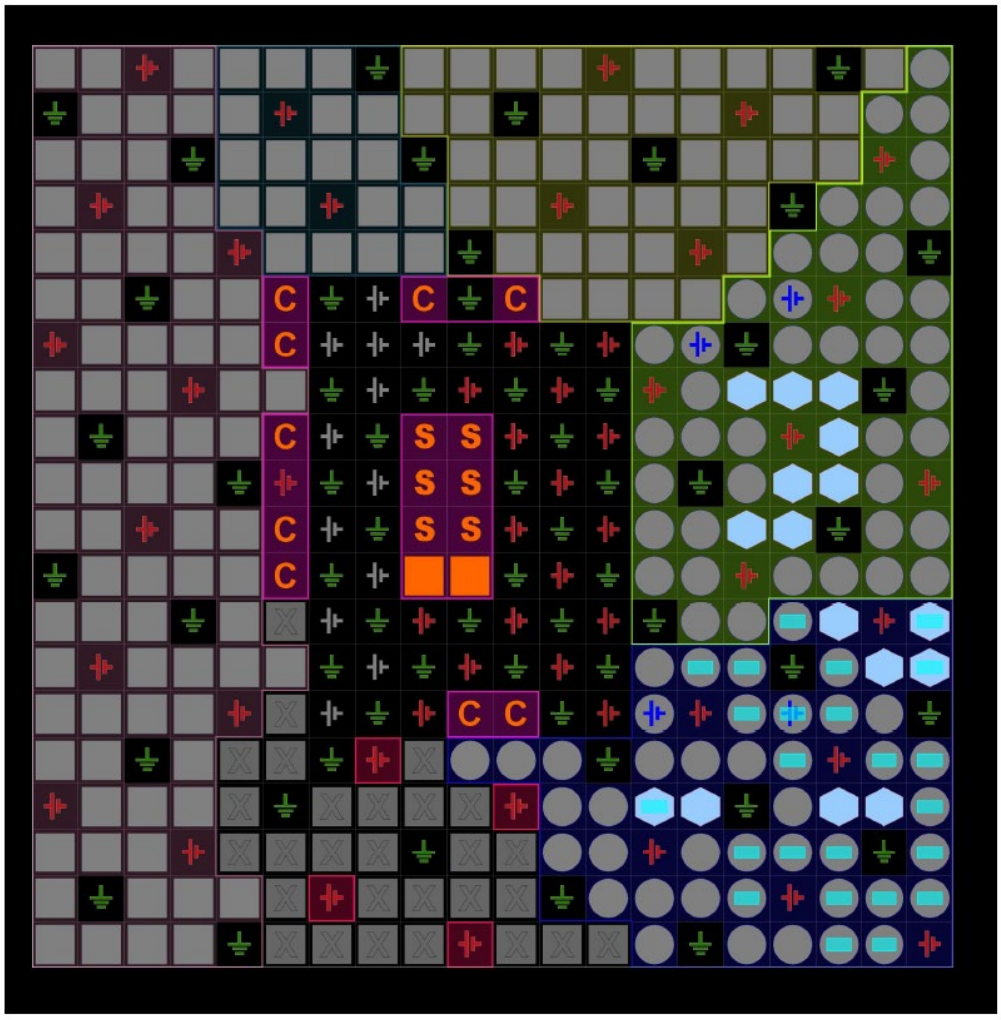


Figure 22. IO Planning

Annexes

Table 2. Timing Report Summary

Slice Logic

Site Type	Used	Fixed	Available	Util%
Slice LUTs	285	0	17600	1.62
LUT as Logic	285	0	17600	1.62
LUT as Memory	0	0	6000	0.00
Slice Registers	83	0	35200	0.24
Register as Flip Flop	83	0	35200	0.24
Register as Latch	0	0	35200	0.00
F7 Muxes	34	0	8800	0.39
F8 Muxes	8	0	4400	0.18

Memory

Site Type	Used	Fixed	Available	Util%
Block RAM Tile	0	0	60	0.00
RAMB36/FIFO*	0	0	60	0.00
RAMB18	0	0	120	0.00

DSP

Site Type	Used	Fixed	Available	Util%
DSPs	0	0	80	0.00

Clocking

Site Type	Used	Fixed	Available	Util%
BUFGCTRL	1	0	32	3.13
BUFIO	0	0	8	0.00
MMCME2_ADV	0	0	2	0.00
PLLE2_ADV	0	0	2	0.00
BUFMRCE	0	0	4	0.00
BUFHCE	0	0	48	0.00
BUFR	0	0	8	0.00

Specific Feature

Site Type	Used	Fixed	Available	Util%
BSCANE2	0	0	4	0.00
CAPTUREE2	0	0	1	0.00
DNA_PORT	0	0	1	0.00
EFUSE_USR	0	0	1	0.00
FRAME_ECCE2	0	0	1	0.00
ICAPE2	0	0	2	0.00
STARTUPE2	0	0	1	0.00
XADC	0	0	1	0.00

Primitives

Ref Name	Used	Functional Category
LUT6	174	LUT
FDCE	83	Flop & Latch
LUT5	68	LUT
LUT4	35	LUT
MUXF7	34	MuxFx
LUT2	33	LUT
LUT3	19	LUT
IBUF	16	IO
CARRY4	13	CarryLogic
OBUF	8	IO
MUXF8	8	MuxFx
LUT1	2	LUT
BUFG	1	Clock