FPGA Prototyping of OFDM transmitter using HDL coder for MmWave communication

A Project Report

submitted by

GUTHULA SATHISH

in partial fulfilment of the requirements
for the award of the degree of

MASTER OF TECHNOLOGY

DEPARTMENT OF ELECTRICAL ENGINEERING
INDIAN INSTITUTE OF TECHNOLOGY MADRAS.

June 2017

THESIS CERTIFICATE

This is to certify that the thesis titled **FPGA prototyping of OFDM transmitter using HDL coderfor MmWave communication**, submitted by **GUTHULA SATHISH** bearing Roll No: **EE15M067**, to the Indian Institute of Technology, Madras, for the award of the degree of **Master of Technology**, is a bonafide record of the research work done by him under our supervision. The contents of this thesis, in full or in parts, have not been submitted to any other Institute or University for the award of any degree or diploma.

Prof. Radha Krishna Ganti
Research Guide
Asst. Professor
Dept. of Electrical Engineering
IIT-Madras, 600 036

Place: Chennai

Date: 9th June, 2017

ACKNOWLEDGEMENTS

I would like to express my sincere thanks to my project guide, Dr.Radhakrishna Ganti, for all the helps and facilities provided to make this project a reality. His wholehearted encouragement, trust and guidelines helped me in the successful completion of this project. I am privileged to be a student at IIT Madras and I express my sincere gratitude to all the teachers for all the academic insights I obtained from them. With their guidance, I gained some wonderful insights into the field of communication systems and into the research environment in general.

I would like to thank all my friends at IIT for their support and and encouragement that helped me to keep myself occupied.Especially I would like to thank Zaid who helped me in vivado design. I express my thanks to my lab mates Manoj Kumar, Ravi and all others for maintaining a cheerful atmosphere

My deepest gratitude to my mother and father for their tremendous amount of support, encouragement, patience, and prayers.

ABSTRACT

KEY WORDS: OFDM(orthogonal frequency division multiplexing),FPGA(Field Programmable Gate Array),HDL (Hardware Description Language),Simulink ,HDL Coder,Virtex 7, FMC230(DAC),FMC126(ADC)

OFDM Transmitter is implemented on FPGA (Virtex 7) , the digital processing part is done in FPGA and these digital signal are converted into Analog using FMC230. Output of DAC is a baseband signal of one sided bandwidth 360MHz,720MHz.The Transmitter algorithms are written in simulink and this simulink model is used to generate HDL code .

The area and timing optimisation of the fixed point design is done with the aid of traceability reports generated by the HDL coder. Timing optimisation is done by Distributed pipe lining and area optimisation is done by resource sharing and word length optimisation. The generated HDL code is verified using FPGA-in-the-Loop cosimulation by interfacing Simulink with FPGA board Xilinx virtex-7 .The rapid FPGA prototyping of OFDM Transmitter using High Level Synthesis tool such as Simulink and the use of High speed DAC and ADC and modifying there reference designs and verification challenges are the main focus of the thesis.

TABLE OF CONTENTS

LIST OF FIGURES

CHAPTER 1

Introduction

Orthogonal frequency division multiplexing (OFDM) is used widely in the current wireless communication system because of its high data rate transmission and the robustness against frequency selective fading.OFDM converts frequency selective fading channel to set of parallel flat fading channels.

In this Project I have used used OFDM modulation Scheme to achieve high data rate.I have used HDL coder to generate HDL code . I have generated OFDM IP from HDL coder and used the same in vivado.

For Digital to Analog conversion we used FMC230 which can operate a 2.4 GSps to get high data rate.In depth understanding of FMC230 is provided in chapter 4.

For Analog to Digital conversion we used FMC126 which can operate a 1.25 GSps to get high data rate.In depth understanding of FMC126 is provided in chapter 5.

I have used matlab HDL verifier to check the data validity and to verify the algorithms.The process of generating HDL code from HDL coder is described in chapter 6.

CHAPTER 2

OFDM Modulation and Demodulation

2.1 Transmitter Architecture:

Below Diagram is the Transmitter Architecture

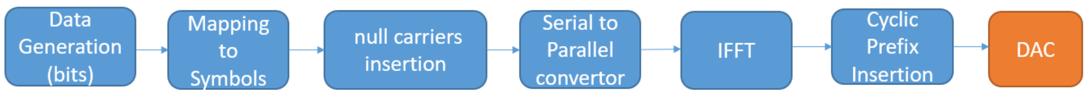


Figure 2.1: Transmitter Architecture

Data Generation: We generate binary data using LFSR or else we can give known data bits.

Mapping to Symbols: The generated bits are mapped to symbols using only one of the modulation schemes such as QPSK,16-QAM etc.

Null carrier insertion: We insert zeros at the both ends to prevent spectral growth and a zero at middle to avoid saturation at amplifier.

serial to parallel convertor: We convert the serial data to parallel and give it to IFFT block.

IFFT: This block applies IFFT on the data which is the major operation in OFDM modulation.

Cyclic Prefix Insertion: Here a part from the rear part of symbol is copied on the front of the symbol. It is useful to mitigate multipath effect and it can be used to detect symbol timing and frequency offset correction.

DAC: All the digital signals that come out of above block are converted to analog signals by using FMC230 (DAC).

2.2 Receiver Architecture

Below is the receiver architecture.

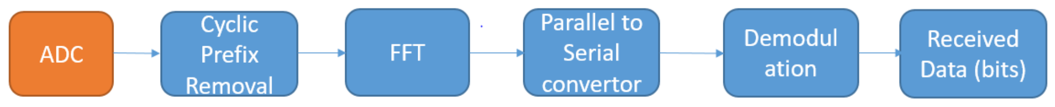


Figure 2.2: Receiver Architecture

Above Receiver Architecture is the basic Receiver architecture in which the analog signal is converted to digital using ADC (FMC126) and these digital and signals are fed into cyclic removal block in which cyclic prefix is removed and after that we apply FFT operation on the signal. Later we demodulate the signal to extract the bits.

CHAPTER 3

OFDM Simulink Model

3.1 Trasmitter:

3.1.1 Source:

Below is the simulink model for source.In this section random bits are generate and these bits are mapped to symbols usinq QPSK modulaion.

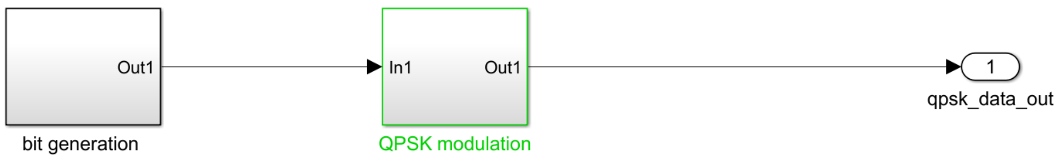


Figure 3.1: QPSK symbol generation

3.1.2 Null carrier insertion:

Simulink model for Null carrier insertion is shown below. The IFFT_Subcarrier_Counter generates a counter signal in the range of 0 to IFFT Length, where each value corresponds to one IFFT bin (or OFDM subcarrier). The remainder of the logic in the top-level of the OFDM Symbol Mapping block maps the data to the central subcarriers of the IFFT, zeros the DC (zero frequency) subcarrier, and reserves the correct number of samples for the CP extension operation. A valid out signal is also generated to indicate the validity of data further down the signal processing path. As there is a pipeline delay associated with the symbol mapping stage, the valid signal is therefore delayed to match the pipeline delay of the data path.

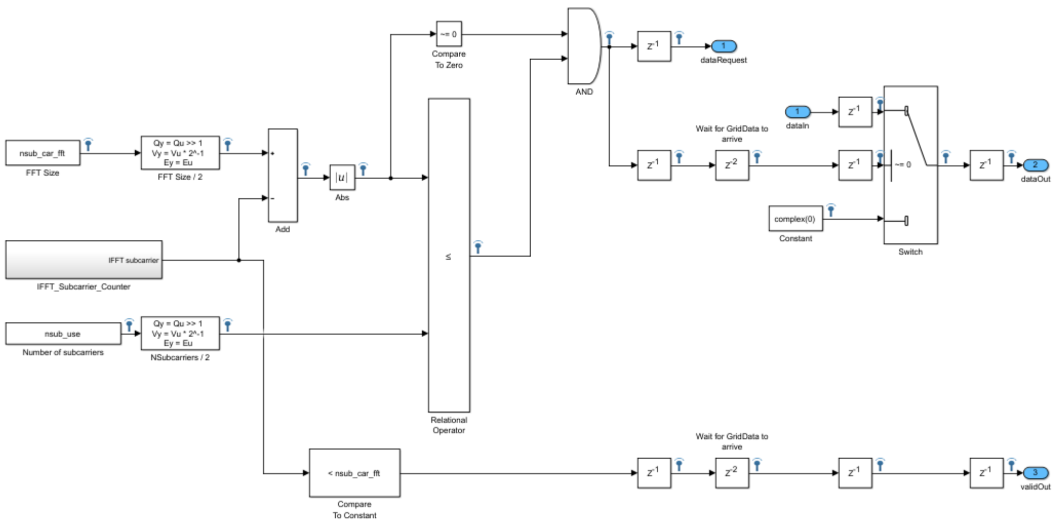


Figure 3.2: Null carrier insertion

Frame Structure:

-255	**-254**	**------**	**-----**	**-----**	**0**	**1**	**-----**	**------**	**-----**	**256**

Figure 3.3: Symbol Structure

3.1.3 FFT Shift:

Above structure is shifted as follows using FFTSHIFT.The incoming OFDM symbol samples are written to a Dual Port RAM block with an addressable range of 2 x IFFT size. After an initial latency of one IFFT frame, the first FFT shifted samples can be read from RAM. The correct read address for the FFT shifted samples is computed by the Compute_Read_Address subsystem. As there is a single clock cycle delay associated with reading data from the RAM block, the valid signal is therefore delayed accordingly.

0	**1**	**------**	**-----**	**-----**	**256**	**-1**	**-----**	**------**	**-----**	**-255**

Figure 3.4: Symbol Structure after shifting

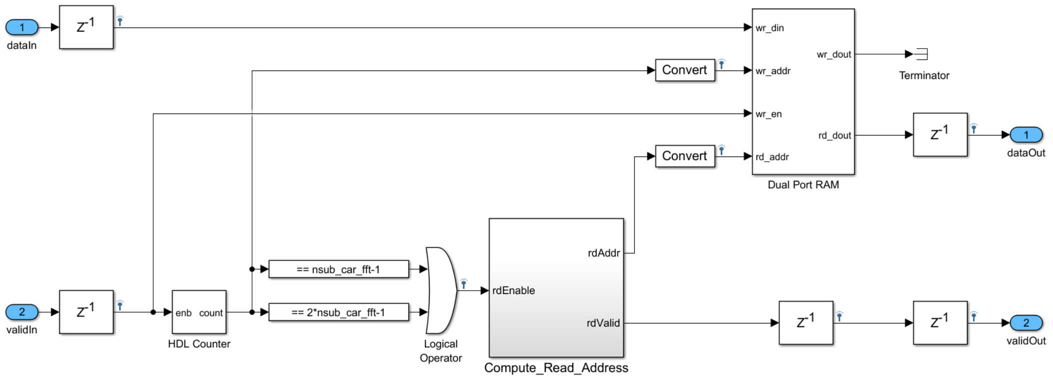


Figure 3.5: FFT SHIFT

3.1.4 Cyclic Prefix Insertion:

The Compute_CP_Pass_Through_Index subsystem detects the end of the current OFDM symbol, determines the length of the CP that corresponds to that symbol and subtracts it from the IFFT length to create the fftMinusCP output. This value is used to schedule the CP extension. Another output, fftMinusCPandWindow, is created and used to schedule the windowing operations. The Check Window Multiply subsystem schedules the windowing operation by creating control and indexing signals, which are used by the Multiply_by_Window subsystem to multiply the "head" and "tail" of the OFDM symbol by raised-cosine windowing samples.

Compute_CP_Pass_Through_Index: As the length of the CP is longer for the first symbol in each slot, it is necessary to detect the end of each OFDM symbol. The end of the symbol can be determined by the change in level of the validIn signal. The detection logic creates a strobe which is high for a single clock cycle which, in turn, enables the OFDM Symbol Counter to advance.

Multiply_by_Window: The windowing process is split into two subprocesses (one for the head of the symbol, and one for the tail) to allow a windowing length that is greater than that of the CP. If the windowing length were to be constrained to less than or equal to that of the CP, this process could be optimized to use a single counter and Lookup Table (LUT) combination. The blocks which multiply the head of the OFDM symbol are situated at the top of the subsystem. As cyclic prefix extension is required for an OFDM modulator, depending on the window length, the windowing of the head may be applied to only the CP samples, and not those that make up the symbol itself. For the relevant CP data samples (0 to window length), an up-counter is used to address a Lookup Table (LUT) containing the pre-calculated Raised-Cosine windowing samples. The output of the RC Window Lookup is then multiplied by the incoming CP data sample. All non-windowed samples are multiplied by 1. The blocks which perform the window multiplication on the tail of the OFDM symbol are situated in the bottom half of the subsystem. In a similar fashion to the windowing of the head samples, for the relevant tail samples (the final IFFT length - window length-1 to IFFT length-1 samples), a down-counter addresses an LUT containing the windowing samples. The output of the RC Window Lookup is then multiplied by the incoming CP data sample. All non-windowed samples are multiplied by 1.

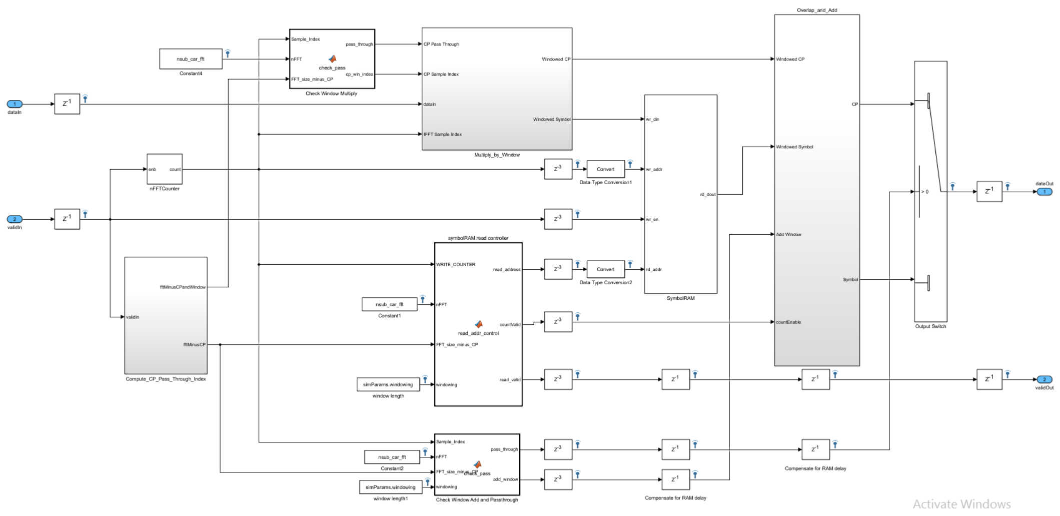


Figure 3.6: Cp insertion

CHAPTER 4

Digital to Analog Convertor (FMC230)

4.1 Introduction

FCM230 is a Dual channel 14-bit D/A convertor , it operates at 5.6Gsps (provided external clock). With internal clock each channel operates at 2.5Gsps. It has features like Burst control, Number of bursts per second and Burst size which can be configured through software. There is one trigger input for customized sampling control. The FMC daughter card is mechanically and electrically compliant to FMC standard (ANSI/VITA 57.1). The FMC has a high-pin count connector, front panel I/O, and can be used in a conduction-cooled environment.

4.2 Characteristics of DAC

Number of channels : 2

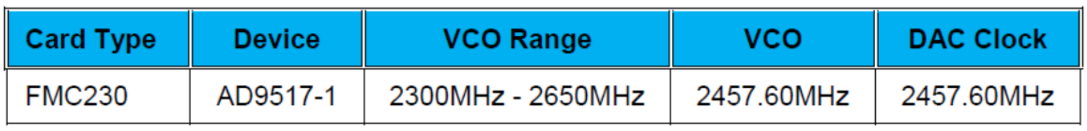
Channel Resolution : 14-bit

Output voltage range 1.12 Vpp (5 dBm)

Output impedance : 50Ω (optimized output impedance for mixed-mode operation)

Analogue output bandwidth : 1.4GHz

4.2.1 Clock characteristics



Card Type	Device	VCO Range	VCO	DAC Clock
FMC230	AD9517-1	2300MHz - 2650MHz	2457.60MHz	2457.60MHz

Figure 4.1: Clock configuration

4.3 Firmware Architecture

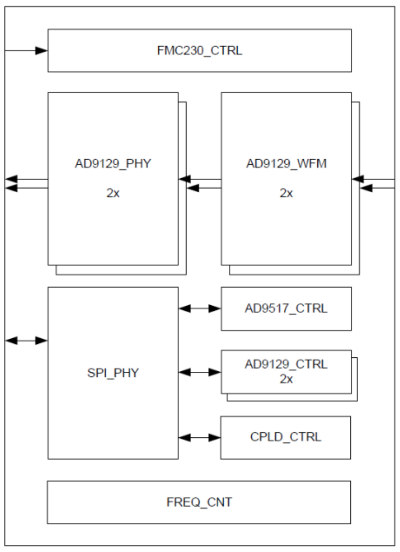


Figure 4.2: Firmware Archicture

4.4 4DSP Reference Firmware Architecture

FMC230 is a product of 4DSP, they have there own tool known as Stellar IP. They have created architecture for FMC230 using Stellar IP tool which can't understand VHDL or Verilog.In this architecture each block is given a name called star and each star is connected by a interface known as wormhole.This Reference Frimware has following Stars.

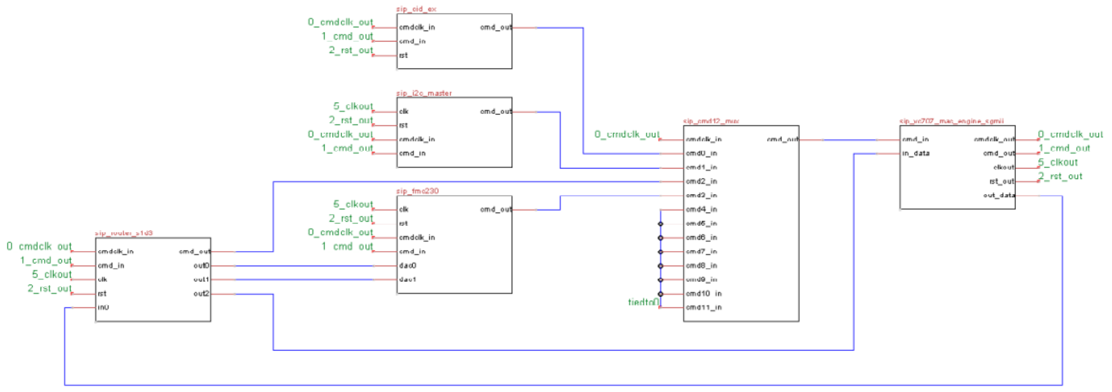


Figure 4.3: FMC230 Fimware

The constellation has the following stars:

1. Constellation ID Star (sip_cid)
2. MAC Engine Star (sip_vc707_mac_engine_sgmii)
3. 1-to-3 Router Star (sip_router_s1d3)
4. FMC230 Star (sip_fmc230)
5. I2C Master Star (sip_i2c_master)
6. Command Multiplexer Star (sip_cmd12_mux)

This Firmware takes data from PC through Ethernet and convert that data to Analog waveform. This Firmware should be modified according to our own requirement.Among above mentioned Stars the one that is useful to us is FMC230 Star(sip_fmc230).

4.4.1 FMC230 Star (sip_fmc230) Description

The FMC230 star takes care of communication with the FMC230 daughter card. Furthermore, it sends data to the D/A converters and provides burst size control. The FMC230 star has an internal waveform memory (WFM) in the D/A path to accommodate the high bandwidth.Below is the FMC230 star

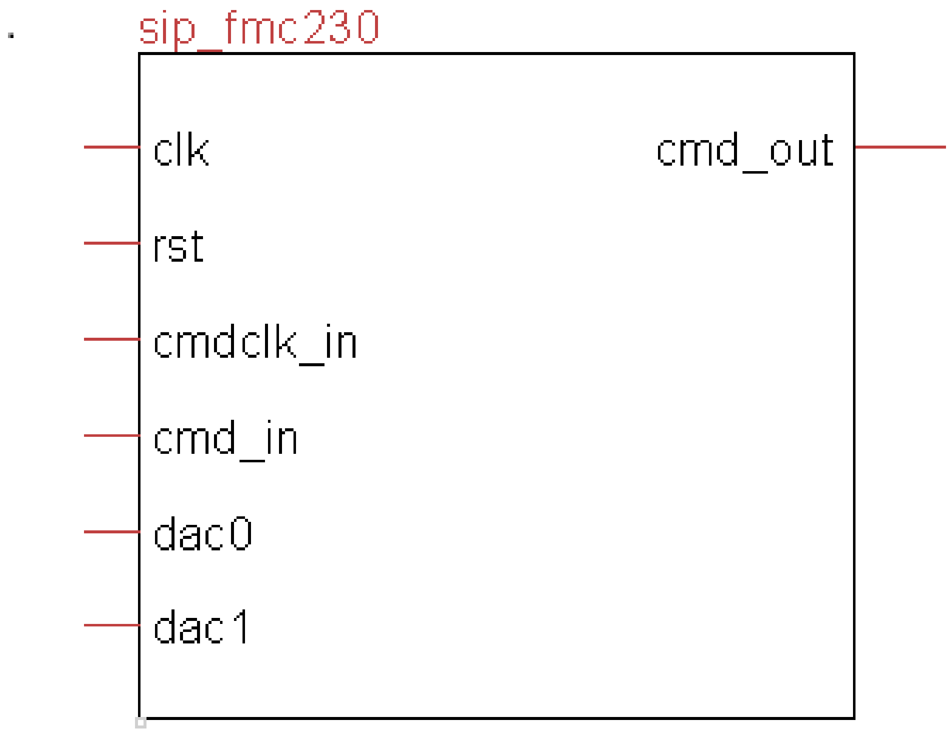


Figure 4.4: FMC230 star

This Star has Stellar IP interface known as wormhole, this wormhole interface is not compatible with our vivado.

clk : clock input to star

rst : reset

cmdclk_in : clock input to command wormhole

cmd_in : command input

dac0 : Dac0 wormhole

dac1 : Dac1 wormhole

We need to convert this Stellar IP interface to Axi interface.

4.5 Conversion of Stellar IP interface to AXI interface

We have to convert the Stellar IP interface(wormhole) to Axi interface , so that we can generate an IP which can be integrated in our own vivado design. We convert cmd_in to axilite interface and Dac interface to axistream interface.

Following is the procedure.

1. Create a new project in vivado 2014.4.
2. Locate the VHDL source of the FMC230 star .It is typically located in the star lib folder "star_lib/sip_fmc230".
3. Copy the IP source files according to the list file "sip_fmc230_v7_vivado.lst" ,this is present in folder "star_lib/sip_fmc230/sip_files".
4. Now we need to copy conversion entities which are useful to convert Stellar IP interface to Axi interface ,these are present at "/4dsp/4FM Core Development Kit/StellarIP/Training Material/AN Materials/AN002/Src/conversion".
5. Copy these files axistream_to_whin, delay_bit, stellarcmd_to_axilite.
6. Copy proper constraint file from "star_lib/sip_fmc230/sip_files".
7. Edit the constraints to meet the timing requirements.
8. Follow AN002 material provided by 4DSP.

All the IP's in sip_fmc230 star were created in vivado 2014.4, it would be better to use vivado 2014.4.

After adding all the files yours sources tab in vivado will have following entities

sip_fmc230, inst_stellarcmd_to_axilite,inst_axistream_to_whin,inst_axistream_to_whin_0

Above entities are to be instantiated under a file axi_fmc230, axi_fmc230 is instantiated by axi_fmc230_synth.

4.6 Vivado Design

All the source files and project that is used to create IP axi_fmc230_synth are present at "E:/FMC230_GOLDEN/FMC230_golden_ip_files"

IP is present at "E:/FMC230_GOLDEN/fmc_230_ip_2016.2"

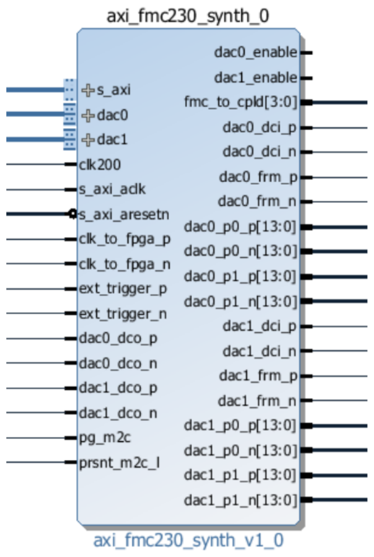


Figure 4.5: FMC230 IP

clk200 should be connected to 200MHz clock, s_axi_aclk should be connected to 125MHz clock,s_axi_aresetn should be connected to suitable reset pin, dac0_enable and dac1_enable are use to enable data source.

s_axi should be connected to microblaze since it is an axilite interface. dac0,dac1 are axi stream interfaces they can be connected to the source which has axistream interface.

Except the above pins remaining pins are to made external which are mapped according to constraints file present in IP itself.

4.7 Usage of Software API

4DSP provide a reference software which is used configure FMC230 through Ethernet, this software takes commands and route them to command input wormhole.We can edit this reference software and use the C++ code on microblaze through SDK.

CHAPTER 5

Analog to Digital Convertor (FMC126)

5.1 Introduction

FMC126 is a four channel 1.25Gsps ADC. It provides four 10-bit ADC channels that enable simultaneous sampling of four channel at 1.25Gsps , two channels at 2.5Gsps and one channel at 5Gsps sample rate.This ADC has 10-bit EV10AQ190 quad 1.25Gsps ADC with DDR LVDS outputs. Below is the block diagram of FMC126.

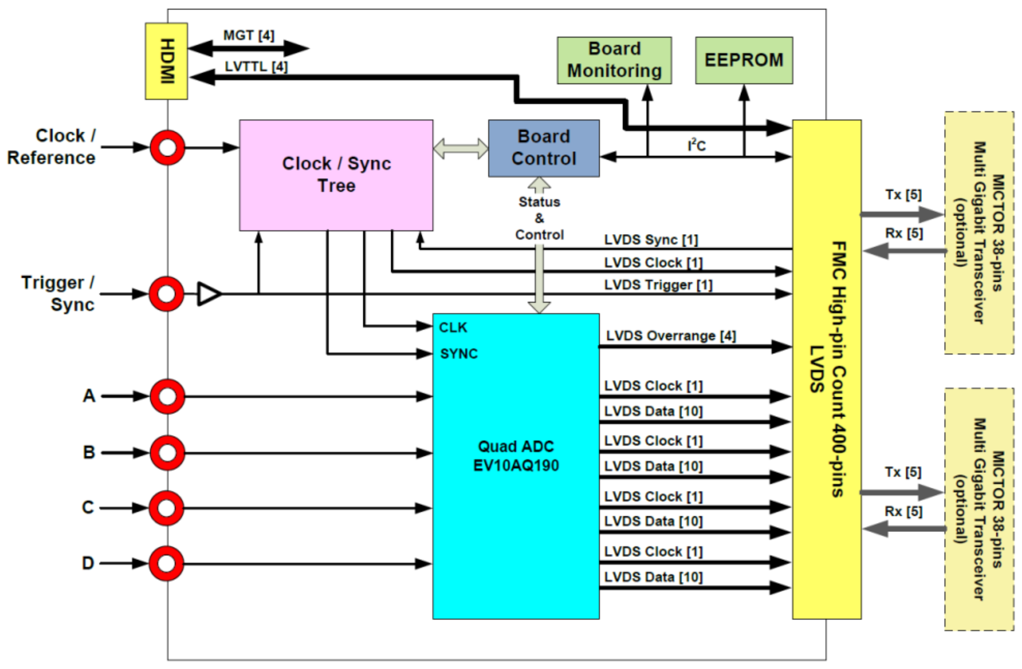


Figure 5.1: FMC126 Block Diagram

5.2 Characteristics of ADC

Number of channels : 1 or 2 or 4

Channel Resolution : 10-bit

Input voltage range: 0.5Vpp

Input impedance : 50Ω

5.3 Output Format

Data is outputted as 16-bit words having the sample bits in the LSB positions. The overrange flag is available in each 16-bit word as well. Unused bits are forced to zeros.

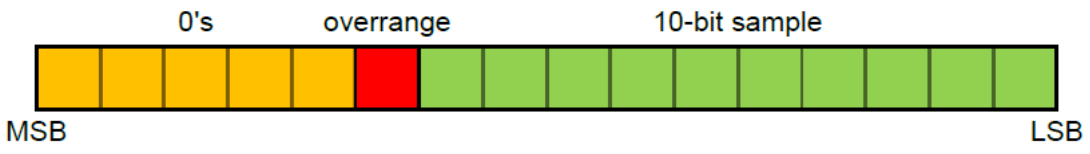


Figure 5.2: Output format of ADC

5.4 4DSP reference Firmware Architecture

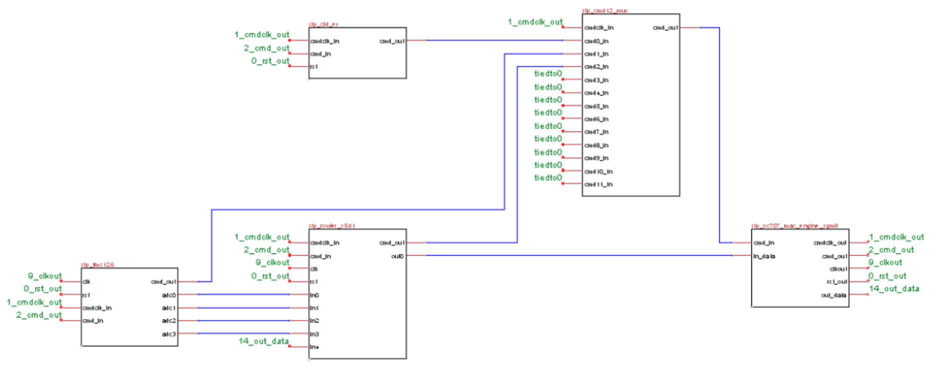


Figure 5.3: Output format of ADC

The constellation has the following stars:

1) Constellation ID Star (sip_cid)

2) MAC Engine Star (sip_mac_engine)

3) 5-to-1 Router Star (sip_router_s5d1)

4) FMC126 Star (sip_fmc126)

5) Command Multiplexer Star (sip_cmd12_mux)

Constellation ID Star:

The constellation ID star holds information about the constellation (constellation ID, star IDs,star address ranges, etcetera). The memory is filled by the Stellar IP tool or manually assigned by the user and can be access by register read cycles

MAC Engine Star:

The MAC engine star distributes register read and register write commands coming from the Ethernet MAC. In addition the star transfers data to/from the host through the Ethernet MAC. This star also generates and distributes the clock and reset signals for the other stars in the constellation.

5-to-1 Router Star:

The 5-to-1 router star routes data from one of the input port to the output port.

FMC126 Star:

The FMC126 star takes care of communication with the FMC126 daughter card and provides burst control. Since the high bandwidth requirement the FMC126 star has internal memories (FIFO) in the A/D path.

Command Multiplexer Star:

The command multiplexer star merges the command output wormholes of all stars and connects a single command output wormhole to the MAC Engine. This star does not require to be controlled by the user.

5.5 Conversion of Stellar IP interface to AXI interface

We have to convert the Stellar IP interface(wormhole) to Axi interface , so that we can generate an IP which can be integrated in our own vivado design. We convert cmd_in to axilite interface and Dac interface to axistream interface.

We can follow the same process mentioned in chapter 4 to build AXI interface.

CHAPTER 6

Generation of HDL Code from Simulink Using HDL Coder

6.1 Introduction to HDL Coder:

HDL coder is a High Level Synthesis tool which can generate VDHL and Verilog codes from MATLAB functions, Simulink models and state flow charts. The generated HDL code can be used for FPGA programming or ASIC prototyping and design.HDL Coder provides a worflow advisor that automates the programming of Xilinx and Altera FPGAs.

In HDL coder,there is a provision to control HDL architecture and implemenation ,highlight critical paths and generate hardware resource utilization estimates. HDL coder provides traceability between the simulink model and the generated VHDl code .

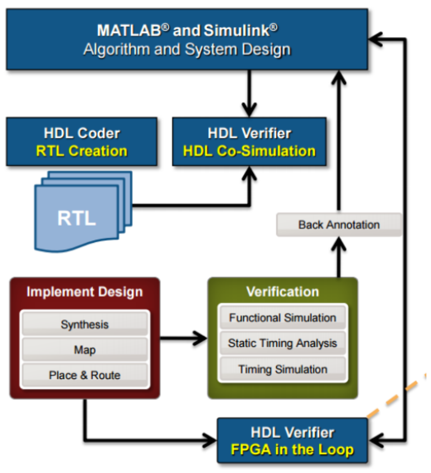


Figure 6.1: Model-Based Design flow using Matlab/Simulink

Code generation using HDL coder is relatively easy with the help of HDL workflow advisor.In this project Simulink modles are used for fPGA prototyping. Above figure describes the complete design flow of HDL code generation and verification using HDL coder and HDl verifier.

6.2 Creating Simulink Model

The first in generation of HDl code starts with creating the simulink model that mimic our algorithm or Algorithm can be directly witten in simulink using blocks that HDL code generation .HDL coder has a library of more than 200 blocks that support HDL code generation which can be used for implementing signal processing algorithms .Basic mathematical operations (addition, multiplication etc.) as well as logical operations can be implemented in the FPGA directly using the blocks from HDL coder library.How ever, complex mathematical operations are difficult to implement in a single step and requires iterative algorithms or mathematical approximations.We can find the block that are supported for HDL genartion by using "hdllib on" command.Below is the HDL simulink library.

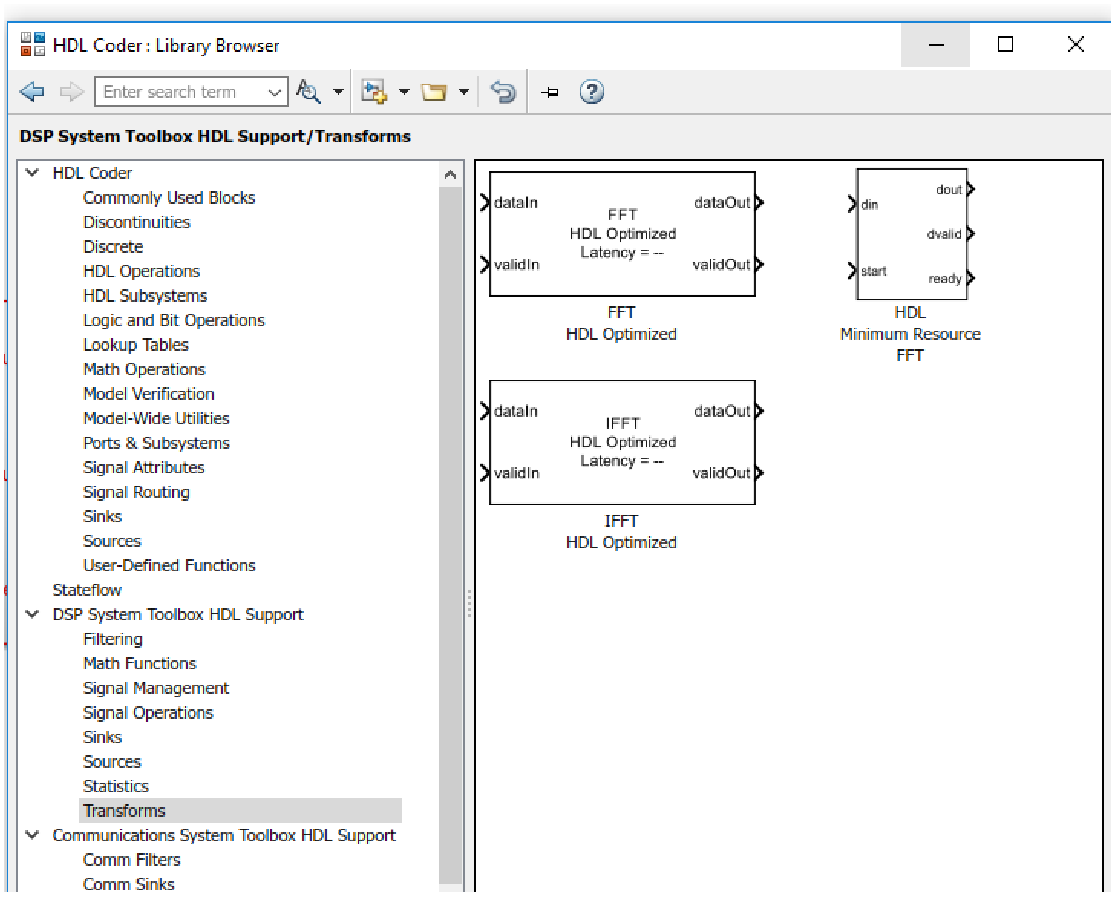
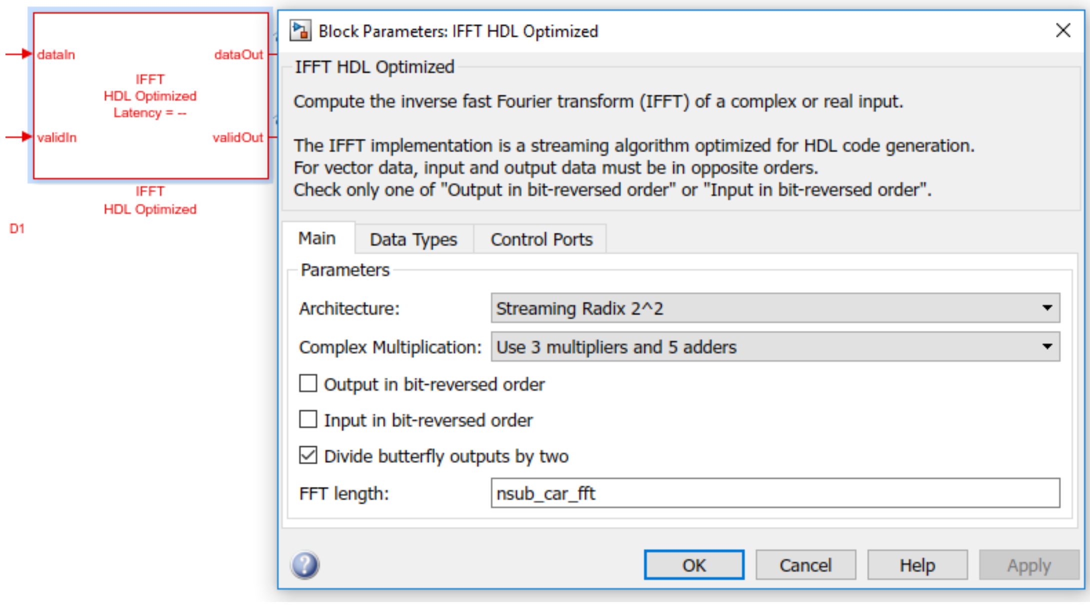


Figure 6.2: SImulink HDL Library



Block Parameters: IFFT HDL Optimized

IFFT HDL Optimized

Compute the inverse fast Fourier transform (IFFT) of a complex or real input.

The IFFT implementation is a streaming algorithm optimized for HDL code generation.
For vector data, input and output data must be in opposite orders.
Check only one of "Output in bit-reversed order" or "Input in bit-reversed order".

Main | Data Types | Control Ports

Parameters

Architecture: Streaming Radix 2^2

Complex Multiplication: Use 3 multipliers and 5 adders

☐ Output in bit-reversed order

☐ Input in bit-reversed order

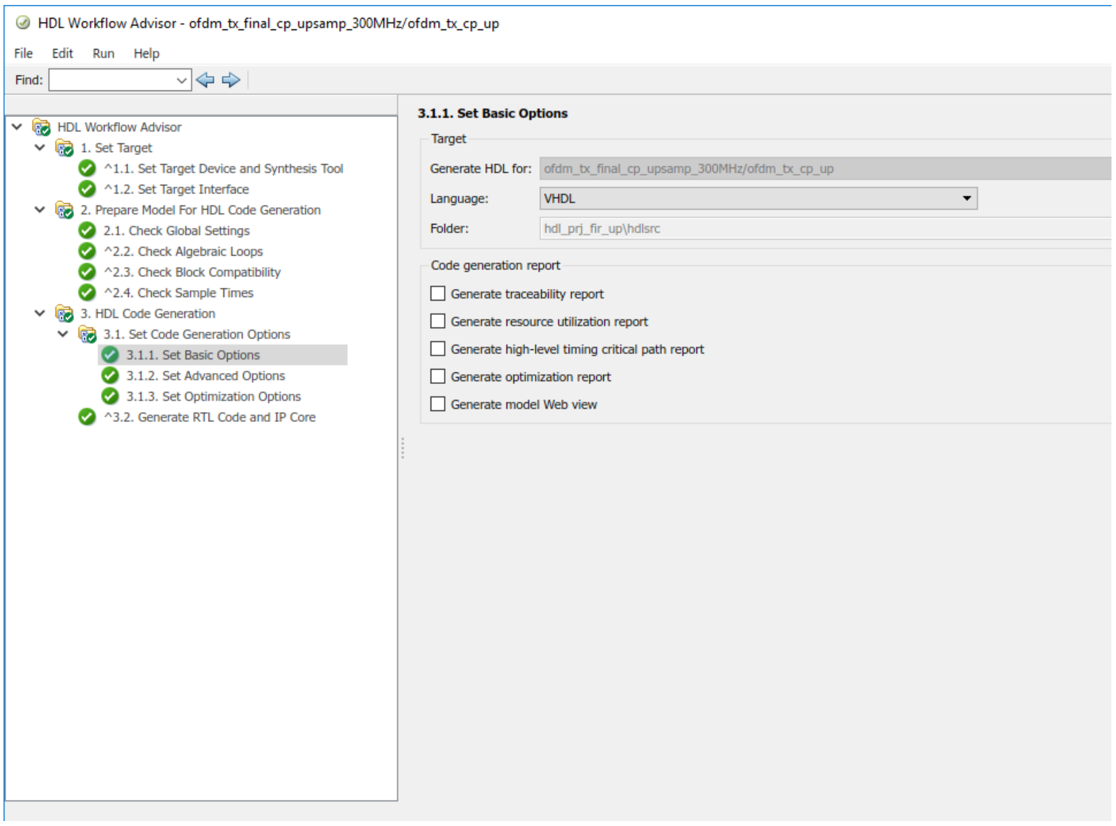
☑ Divide butterfly outputs by two

FFT length: nsub_car_fft

Figure 6.3: FFT block that support HDL generation

6.3 HDL code Generation from Simulink Model

HDl code can be generated from simulink by following step by step procedure laid by HDL workflow advisor. Entire simulink model should be made into a single subsystem and it should be made as atomic before the start of workflow advisor.Below is the worflow advisor screenshot.


HDL Workflow Advisor - ofdm_tx_final_cp_upsamp_300MHz/ofdm_tx_cp_up

- HDL Workflow Advisor
  - 1. Set Target
    - ^1.1. Set Target Device and Synthesis Tool
    - ^1.2. Set Target Interface
  - 2. Prepare Model For HDL Code Generation
    - 2.1. Check Global Settings
    - ^2.2. Check Algebraic Loops
    - ^2.3. Check Block Compatibility
    - ^2.4. Check Sample Times
  - 3. HDL Code Generation
    - 3.1. Set Code Generation Options
      - 3.1.1. Set Basic Options
      - 3.1.2. Set Advanced Options
      - 3.1.3. Set Optimization Options
    - ^3.2. Generate RTL Code and IP Core

3.1.1. Set Basic Options

Target

Generate HDL for: ofdm_tx_final_cp_upsamp_300MHz/ofdm_tx_cp_up

Language: VHDL

Folder: hdl_prj_fir_up\hdlsrc

Code generation report

☐ Generate traceability report

☐ Generate resource utilization report

☐ Generate high-level timing critical path report

☐ Generate optimization report

☐ Generate model Web view

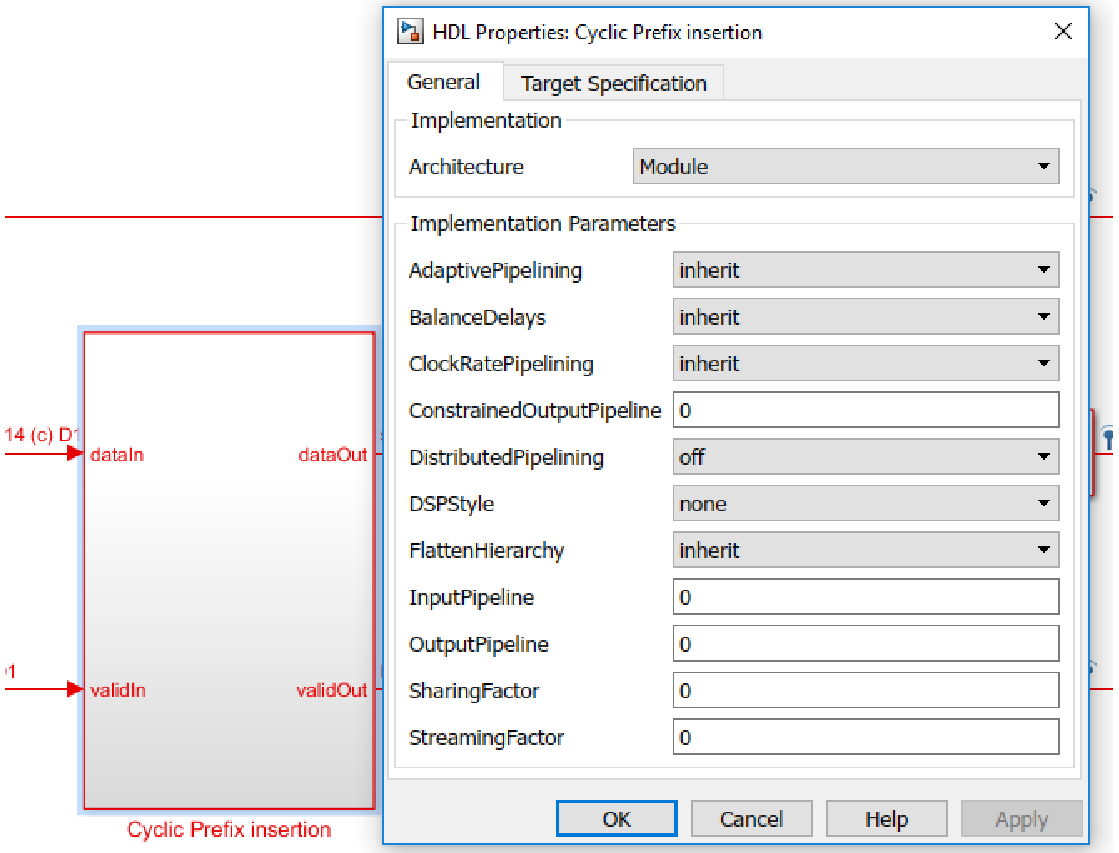
Figure 6.4: Workflow Advisor

There are steps in IP core generation as you can see it from above figure.We need to specify the target interface in section 1.2, it can be AXI Lite, AXi stream or a external port.Once you run section 3.2 our required IP will be generated.In section 3.1 we can set options like traceability report, resource utilization report etc.

6.4 Opimizing Generated HDL Code

After generating HDL code, worflow advisor will generate a report that has estimated critical path. We can reduce this path delay by pipelining that path. We can do pipelining by two ways.

One way is we can directly insert delays on the critical path, each delay will be translated as a register in HDL code. Other way is we can right click the subsystem or block that is in critical path and go to "HDL block properties",there we can insert input and output pipelines.



Implementation

Architecture	Module

Implementation Parameters

AdaptivePipelining	inherit
BalanceDelays	inherit
ClockRatePipelining	inherit
ConstrainedOutputPipeline	0
DistributedPipelining	off
DSPStyle	none
FlattenHierarchy	inherit
InputPipeline	0
OutputPipeline	0
SharingFactor	0
StreamingFactor	0

Figure 6.5: Pipeline

CHAPTER 7

Results

OFDM transmit scheme :

FFT Size	512
Sampling Frequency of DAC	2.4576 GSps
Sub carrier Spacing	4.8MHz
Modulation Scheme	QPSK
No. of Data symbols	150
Occupied Bandwidth (one sided)	360MHz
CP length	40

Output spectrum of OFDM (I component):

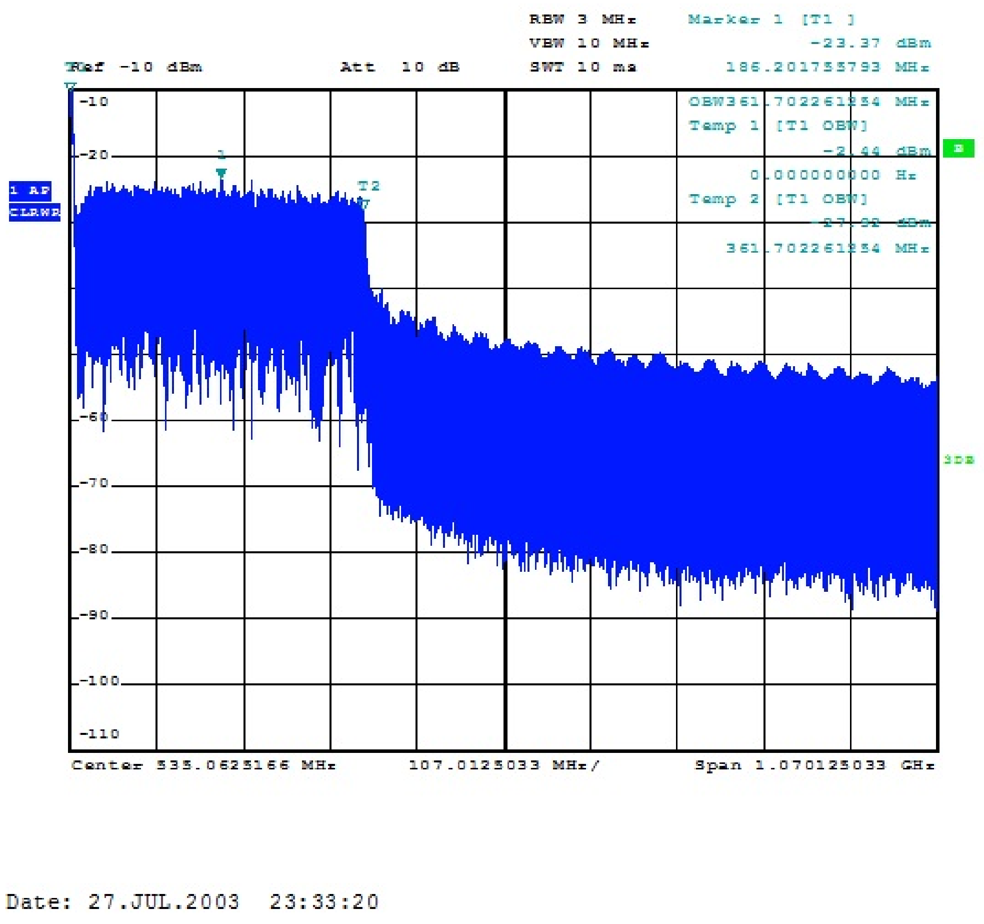


Figure 7.1: I component spectrum

Output spectrum of OFDM (Q component):

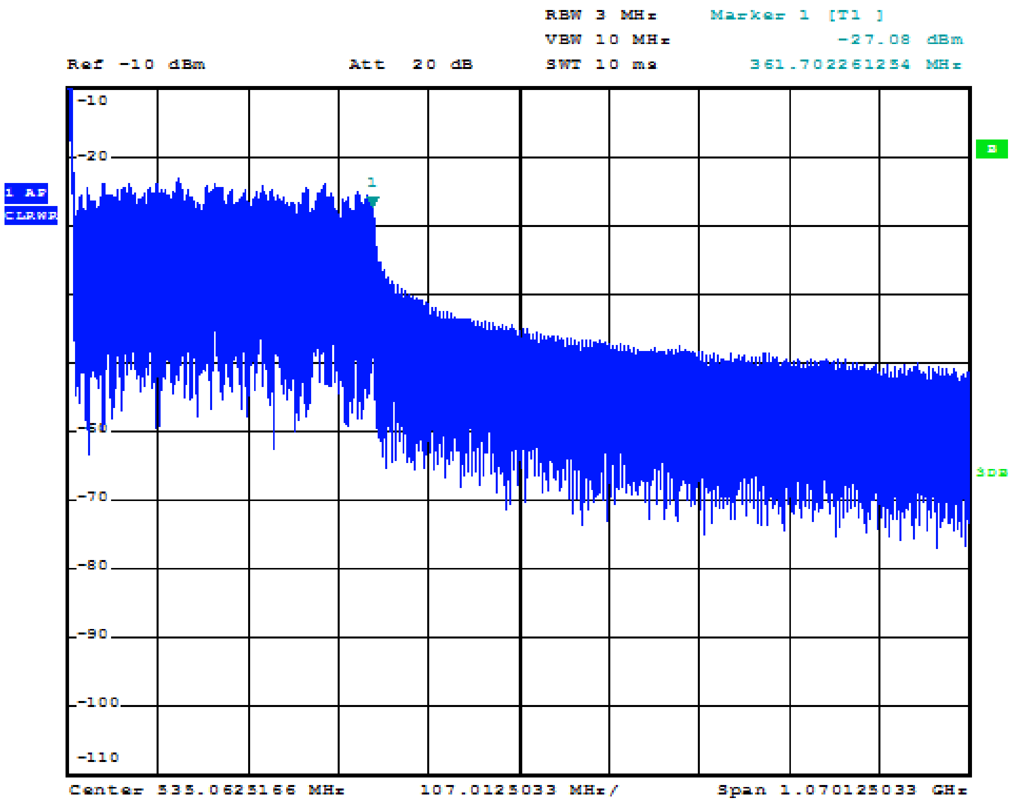


Figure 7.2: Q component spectrum

OFDM transmit scheme :

FFT Size	512
Sampling Frequency of DAC	2.4576 GSps
Sub carrier Spacing	4.8MHz
Modulation Scheme	QPSK
No. of Data symbols	300
Occupied Bandwidth (one sided)	720MHz
CP length	40

Output spectrum of OFDM (I component):

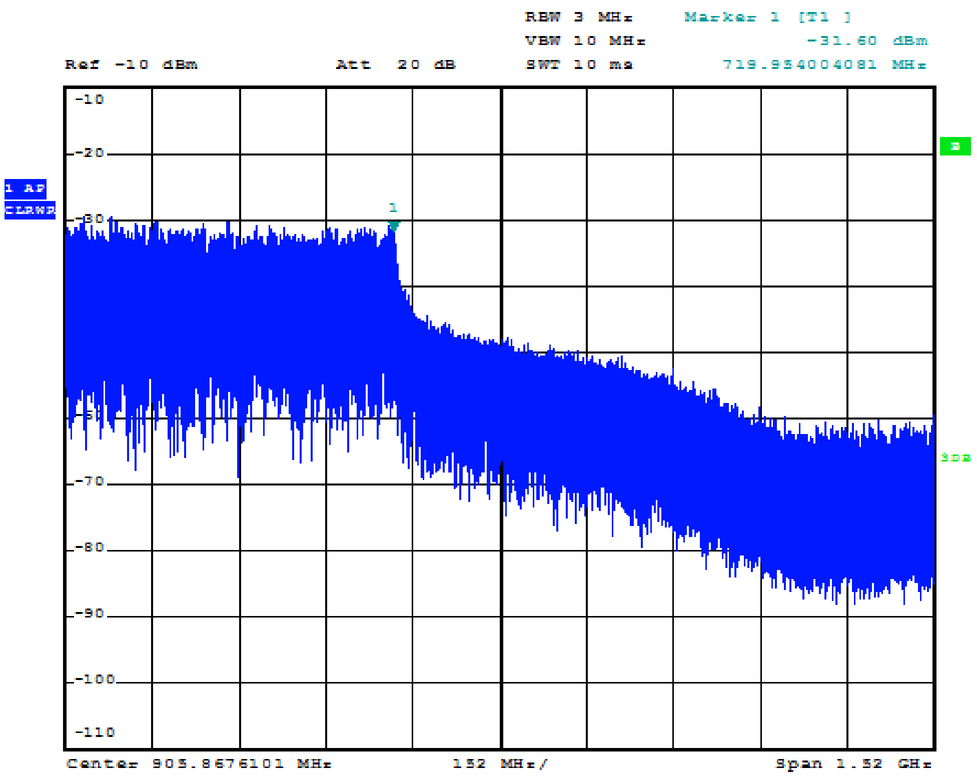


Figure 7.3: I component spectrum

Output spectrum of OFDM (Q component):

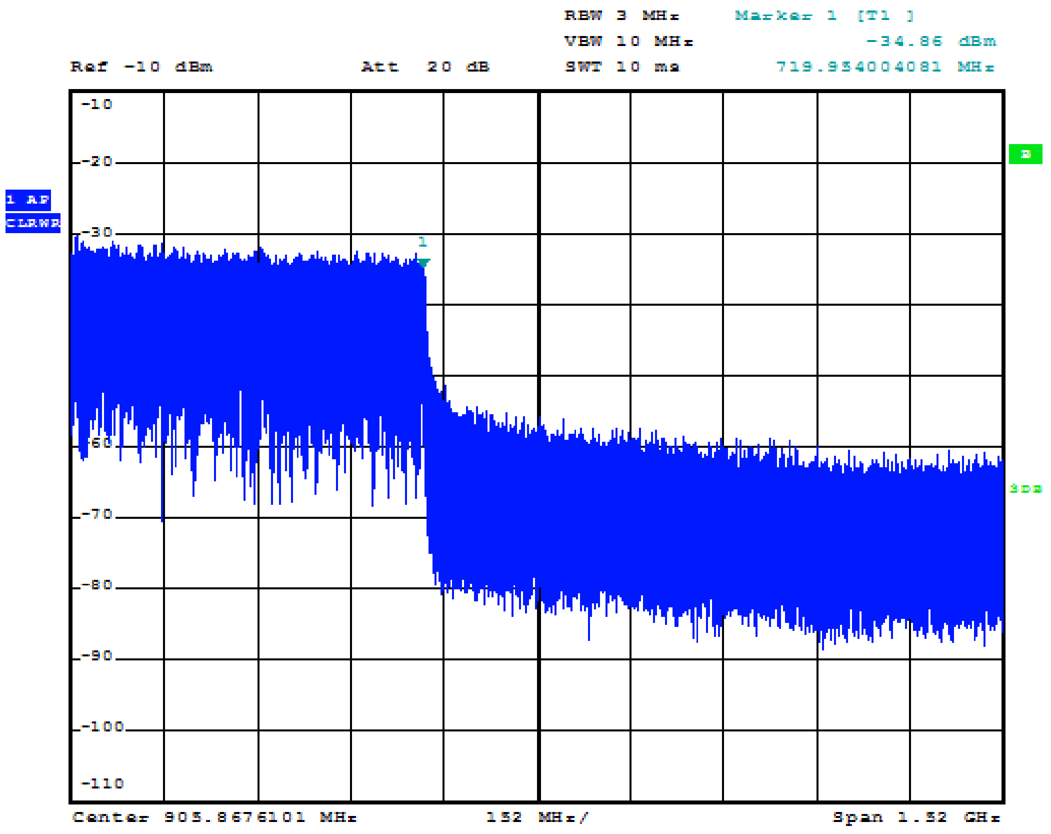
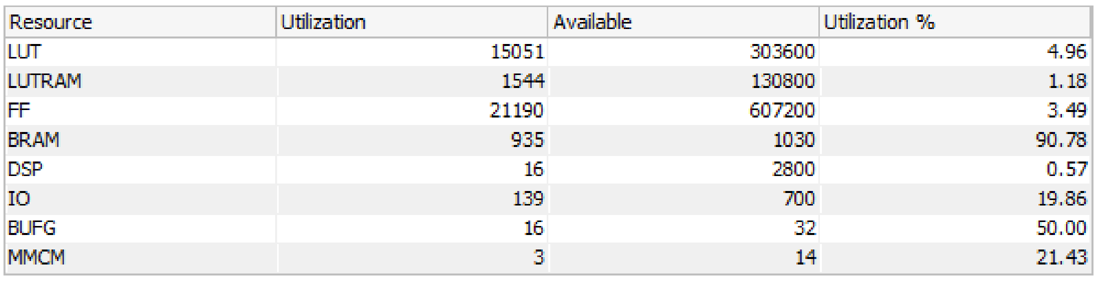


Figure 7.4: Q component spectrum

Resource Utilization Report:



Resource	Utilization	Available	Utilization %
LUT	15051	303600	4.96
LUTRAM	1544	130800	1.18
FF	21190	607200	3.49
BRAM	935	1030	90.78
DSP	16	2800	0.57
IO	139	700	19.86
BUFG	16	32	50.00
MMCM	3	14	21.43

Figure 7.5: Resource Utilization Report