SUMMER INTERNSHIP PROJECT REPORT

Digital Beamforming Techniques for L-Band Satellite using RfSoC FPGA

Submitted by

Pavithra Anil

SC23B115

B.Tech ECE (Avionics)

Indian Institute of Space Science and Technology(IIST)

Under the guidance of

Dr. Rahul G. Waghmare

ACTD Division, Communication Systems Group

U R Rao Satellite Centre (URSC), Bengaluru

July 22, 2025

Contents

1 Abstract

This internship project is focused on designing and implementing a digital beamforming system using a Xilinx Zynq UltraScale+ RFSoC ZCU208 evaluation board for a 4-element linear antenna array. The primary objective was to harness the RFSoC's onboard high-speed ADCs, DACs, and programmable logic to achieve precise beam steering through digital phase-shifting techniques. Some of the key aspects included synchronizing channel signals, developing FPGA-based signal processing algorithms, and validating system performance using simulated and real-time RF signals.The RFSoC platform supported highly integrated processing and low latency, making it well-suited for advanced radar and communication systems.The project successfully demonstrated accurate beam steering in controlled tests, meeting design objectives. This internship provided hands-on experience in digital signal processing, FPGA design, and RF system implementation, fostering technical expertise and practical insights into modern communication technologies.

2 Introduction

Beamforming is a signal processing technique that is used in antenna arrays to control the direction of signal transmission or reception by adjusting the phase and amplitude of signals at each antenna element. It enables spatial filtering, allowing systems to focus on desired signal directions while suppressing interference and noise from others.

In this project, the focus was on implementing digital beamforming using the Xilinx Zynq UltraScale+ RFSoC ZCU208 evaluation board. In digital beamforming, the signals received by each antenna element are digitized using individual independent analog-to-digital converters (ADCs) and combined in the digital domain using software-defined phase shifts. This allows for the formation of multiple simultaneous beams in different directions using the same antenna array. Unlike analog beamforming, which relies on hardware phase shifters and combiners, digital beamforming performs signal processing in the digital domain, offering greater flexibility, precision, and efficiency by replacing physical components with mathematical operations. The RFSoC(Radio Frequency System-on-Chip) platform, with its integrated high-speed ADCs, DACs, and programmable logic, provides a highly efficient solution for real-time multi-channel digital signal processing.

The system was designed for a 4-element linear antenna array, where phase shifts were applied digitally to steer the main beam in desired directions. Key aspects of the project included digital phase alignment, time synchronization, and beam pattern visualization. This report presents the design methodology, hardware implementation, and performance evaluation of the digital beamforming system developed during the internship.

2.1 Objectives

The major objectives of this internship are enlisted as follows:

- To familiarize with the RFSoC ZCU208 development platform and its associated toolchains (Vivado, Vitis).
- To understand and implement signal generation and beamforming using DAC tiles of the RFSoC.
- To configure and test the phase shifting mechanism digitally through Vitis C code.
- To explore methods for phase calibration using oscilloscope measurements.
- To validate beamforming and steering functionality using an anechoic chamber setup.
- To implement beam steering based on satellite position data (e.g., from Orbitron).

3 System Overview

Digital beamforming system consists of Antenna elements arranged in the appropriate array geometry(in this case, linear) ,the RFSoC evaluation board and supporting modules that enable high-speed, multi-channel RF processing.

3.1 ZCU208 RFSoC Evaluation Kit

The core of the system is the **ZCU208 evaluation board**, based on Xilinx's Gen3 Zynq UltraScale+ RFSoC ZU48DR device. The board integrates:

- Real-time processors with quad core CortexA53 and dual CortexR5.
- Eight 14-bit 5 GSPS ADCs and eight 14-bit 10 GSPS DACs capable of direct RF sampling and generation.

- On-chip hard logic supporting eight SD-FEC cores, DDR4 memory, SFP28 links, FMC+ and RFMC2.0 expansion interfaces.

This high-performance RFSoC enables a fully digital signal chain, eliminating many analog RF components and enabling flexible beamforming control.

3.2 CLK104 RF Clock Add-on Card

To achieve accurate sampling and synchronization across multiple DAC/ADC tiles (multi-tile synchronization), we use the **CLK104** clock. Key features:

- Provides ultra-low-noise reference clocks to RF tiles via LMK04828B PLL.
- Supports internal/external clock routing, tile-to-tile synchronization, and inter-board sync capabilities.
- Provides stable clock distribution for RF data conversion, utilizing an RF PLL LMX2594 for ADC clocking with a reference frequency of up to 15 GHz, as specified by Texas Instruments. The CLK-104's precise clocking, combined with the XM655's breakout capabilities, supports robust performance in real-time signal synthesis and beam steering for advanced RF applications.

3.3 XM655 Breakout Add-on Card

For flexible RF interfacing, the **XM655 breakout card** is used. It:

- Routes up to 8 DAC(at 10 GSPS) and 8 ADC(at 5 GSPS) channels from the RFSoC via SMA connectors enabling low-noise signal conversion.
- Supports upto 6GHz operation through selectable baluns and filters over four frequency bands (10 MHz–6 GHz)
- The XM655's balun channels, as detailed in its schematic, ensure high-quality signal acquisition and generation, making it suitable for high-performance RF applications.

This enables custom analog front-end configurations such as filters, amplifiers, or antenna connections.

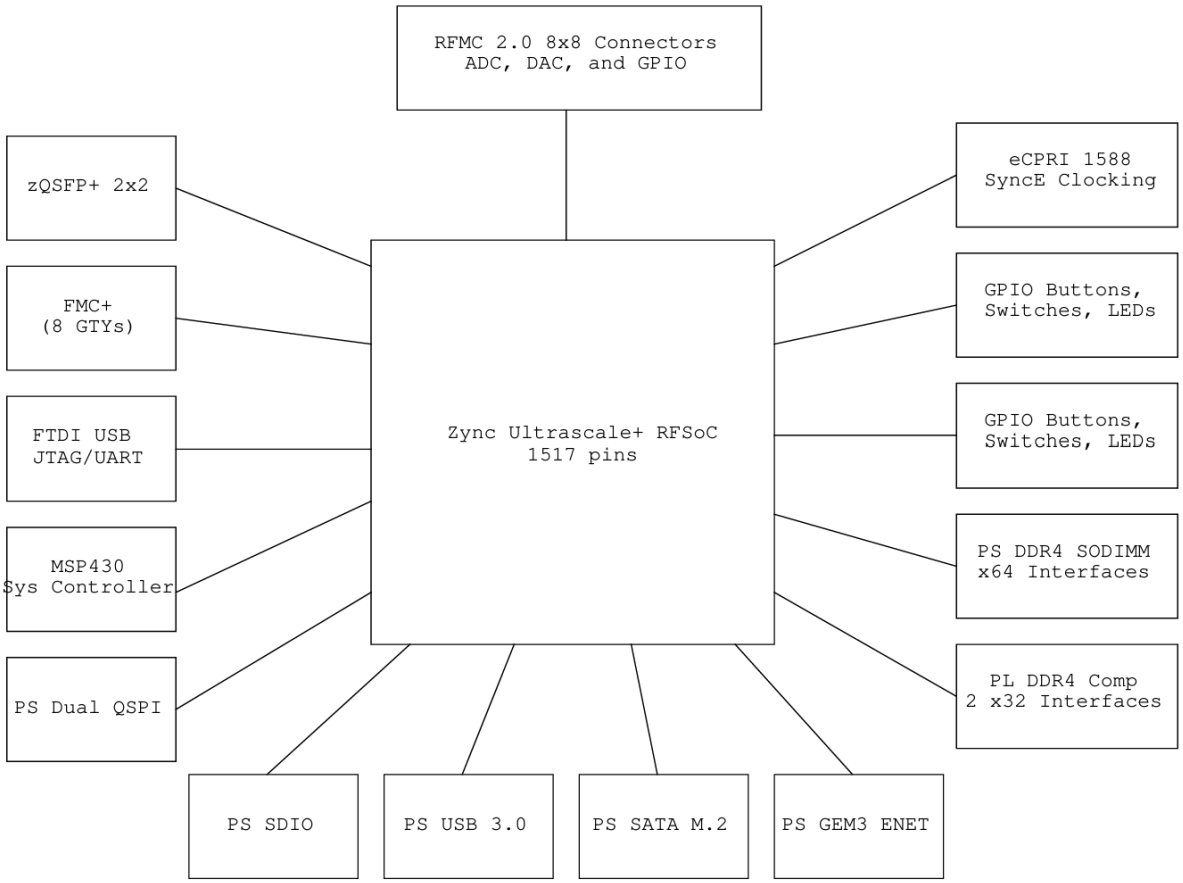


Figure 1: Block diagram of the RFSoC FPGA ZCU208

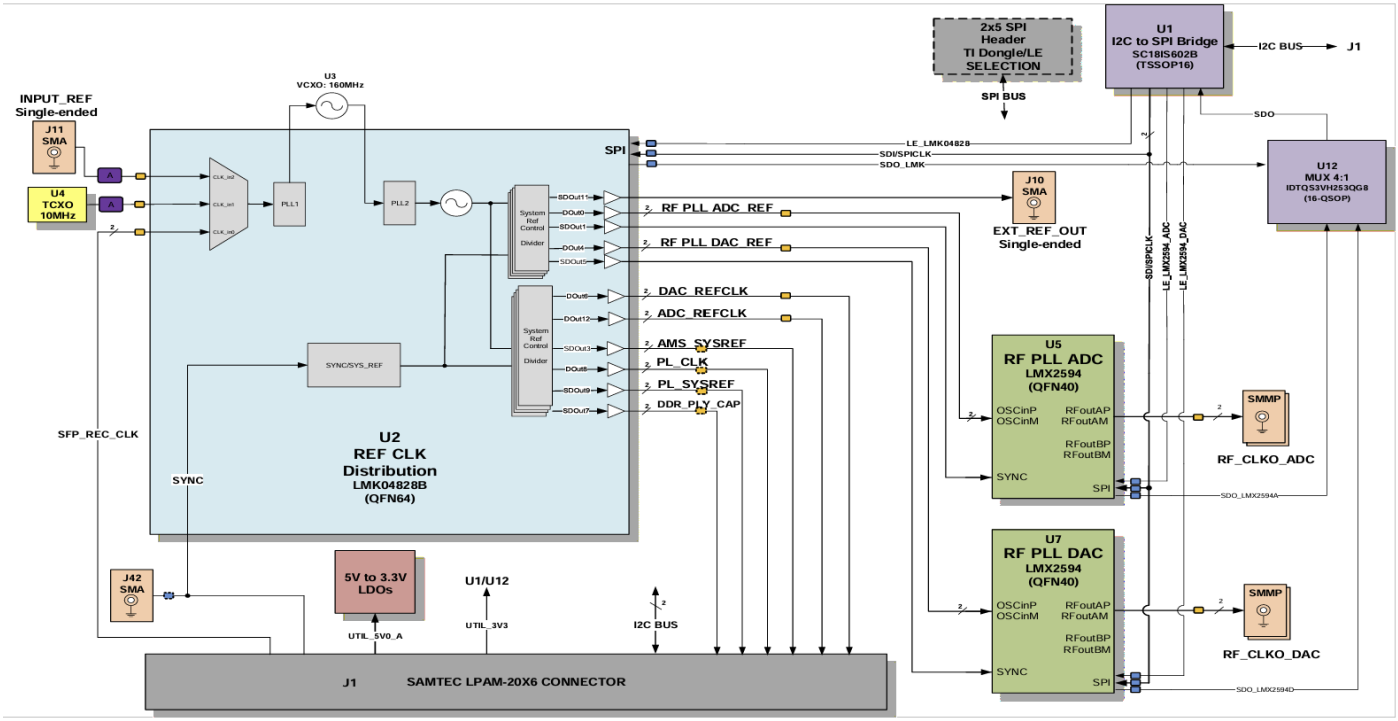


Figure 2: Block diagram of CLK104

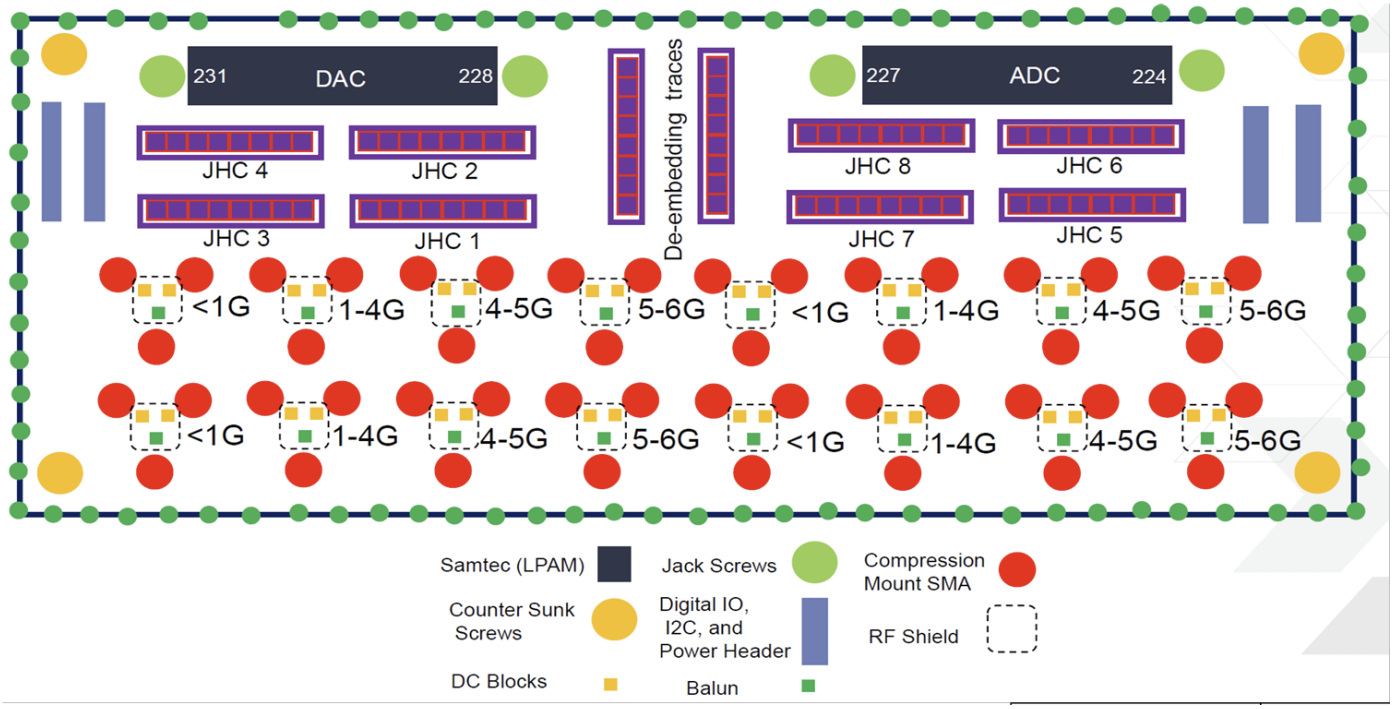


Figure 3: Block diagram of XM655

4 Development of Framework

My aim is to use RF-SoC board for development of an algorithmic framework for DBF. It involves both hardware and software development using the ZCU208 platform. Subsequently, the FPGA board can be used to develop a larger array of antennas. Though the ZCU208 board has 8 nos of RF sampling ADCs, for this project four of them are being used to form a 4 element linear phased array.

4.1 Meteor-M N2-4 HRPT Images

The final objective is to enable onboard and ground receiver antennas to produce several beams based on a common phased array hardware platform. To illustrate the efficiency of the beamforming algorithms implemented, the reception of weather images from the Meteor-M N2-4 satellite has been selected as an indicative case. The Meteor-M series are Russian-owned weather imaging satellites in sun-synchronous polar orbits. These pictures are broadcast on several frequencies with different data quality and levels of compression, offering a perfect environment to test the multi-beam reception ability of the system. The following table gives information regarding the modes and image quality broadcast by the satellite.

Table 1: Modes and quality of the Meteor-M N2-4 transmissions

Sr.No.	Mode	Band	Frequency	data rate	Resolution	Modulation
1	LRPT	VHF	137 MHz	72 kbps	1km	QPSK
2	HRPT	L	1.7 GHz	2.4 Mbps	1km	QPSK

4.2 Overall System Configuration

The digital beamforming system has an antenna element array in a particular geometry(in this case, linear). Each of the antenna elements' received signals is amplified and filtered using a Low Noise Amplifier (LNA) stage. To ensure compatibility and meet the input requirements of the RFSoC's ADCs, the signals are further amplified. The analog signals are converted to digital form by the

RFSoC's ADCs and processed with digital logic realized in the FPGA. This digital processing block executes phase shifting and signal combining on the signals. The beamformed digital signals that result are converted to analog signals again by the RFSoC's DACs. The analog outputs are then routed to the radio receiver to demodulate the signal and decode data. The following block diagram shows the structure of the suggested beamforming system.

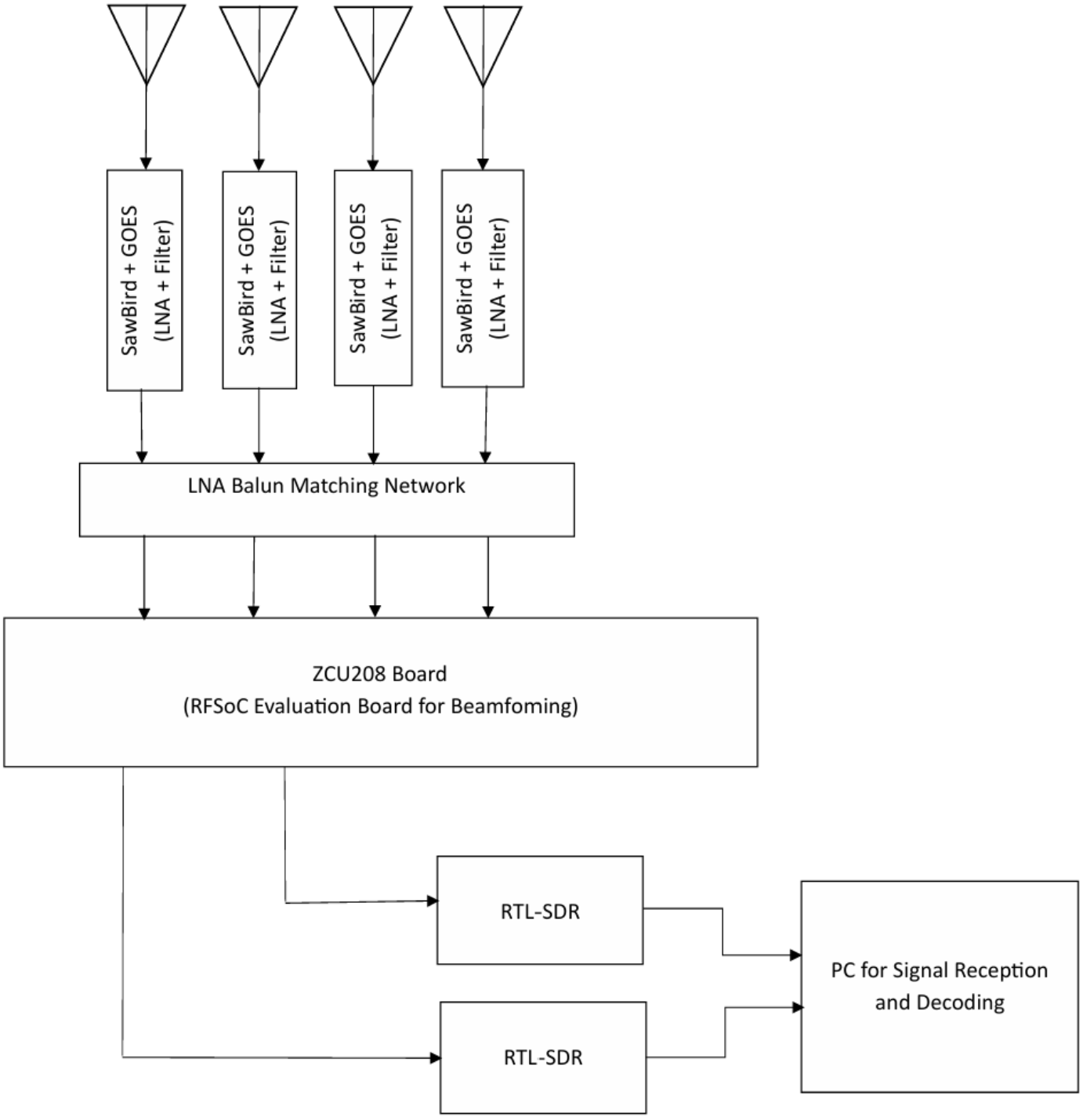


Figure 4: Block Schematic of the digital beamforming PAA

5 Methodology

5.1 Phase Shifter Design and Integration in VHDL

The phase shifter was implemented through a custom hardware block that was designed in Vivado's block design environment. Figure 5 illustrates the complete data path integrated with the Zynq UltraScale+ MPSoC processing system.

The system consists of several floating point operators to carry out the required phase arithmetic, AXI GPIO blocks and AXI-Stream interconnects. The RF signal is filtered through a series of blocks where real-time phase adjustments are made before being directed to the Zynq RF Data Converter core. The VHDL logic for each processing block was managed and parameterized through the software interface using Vitis.

Key functional components include:

- **Multiple PS_Blk Instances**: Each of the four antenna channels uses its own instance of the PS_Blk module for independent beam control.
- **Phase Computation and Control**: Three floating-point adders — **Floating Point 0, 1, and 2** — perform intermediate summations required for computing phase-shifted components across multiple antenna elements.
- **Float-to-Fixed Conversion**: The final computed result is converted back to fixed-point format using **Floating Point 3** before being sent to the Zynq RF Data Converter IP for DAC transmission.
- **GPIO and AXI Interfacing**: AXI GPIO blocks are used to control dynamic phase inputs from the processing system, while the AXI-Stream Interconnect ensures data and control signals are properly routed between the processing blocks.

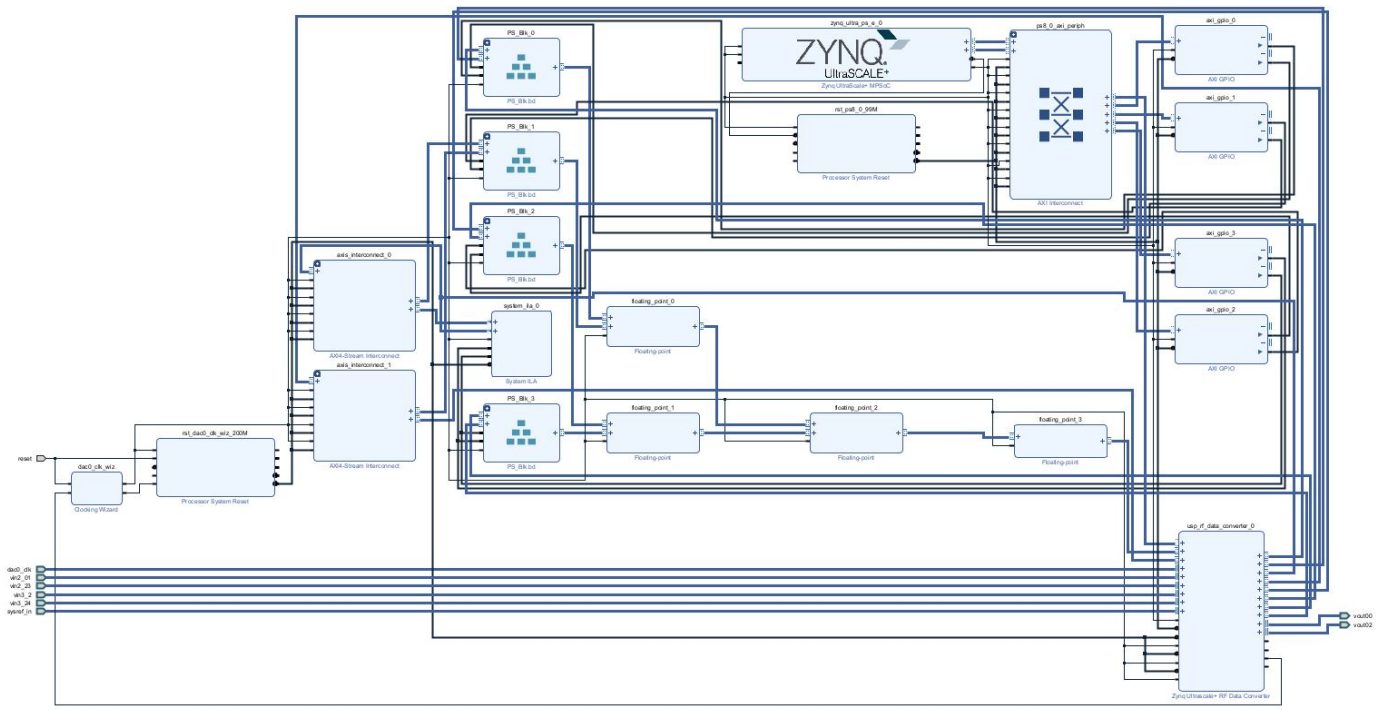


Figure 5: Vivado block design for the digital phase shifter

5.1.1 Internal Design of Phase Shifter Block

Figure 6 shows the internal architecture of each of the PS_Blk modules, which is responsible for applying a phase rotation to complex baseband signals using real-time trigonometric computations. The design takes real (`IData`) and imaginary (`QData`) inputs and performs the following transformation:

$$\text{PSData} = (I \cdot \cos(\theta) - Q \cdot \sin(\theta)) + j(I \cdot \sin(\theta) + Q \cdot \cos(\theta))$$

To realize this:

- **Floating Point 0 and 1**: These blocks perform fixed-to-floating point conversions for the `IData` and `QData` inputs, enabling compatibility with the subsequent arithmetic blocks.
- **Floating Point 2 and 3**: These carry out multiplication operations between the floating point data and trigonometric constants (i.e., $\cos(\theta)$ and $\sin(\theta)$).
- **Floating Point 6**: This final block adds the appropriate results from the multipliers to compute the rotated complex output, `PSData`.

This modular design supports dynamic phase shift control, allowing digital steering of the beam direction in real time.

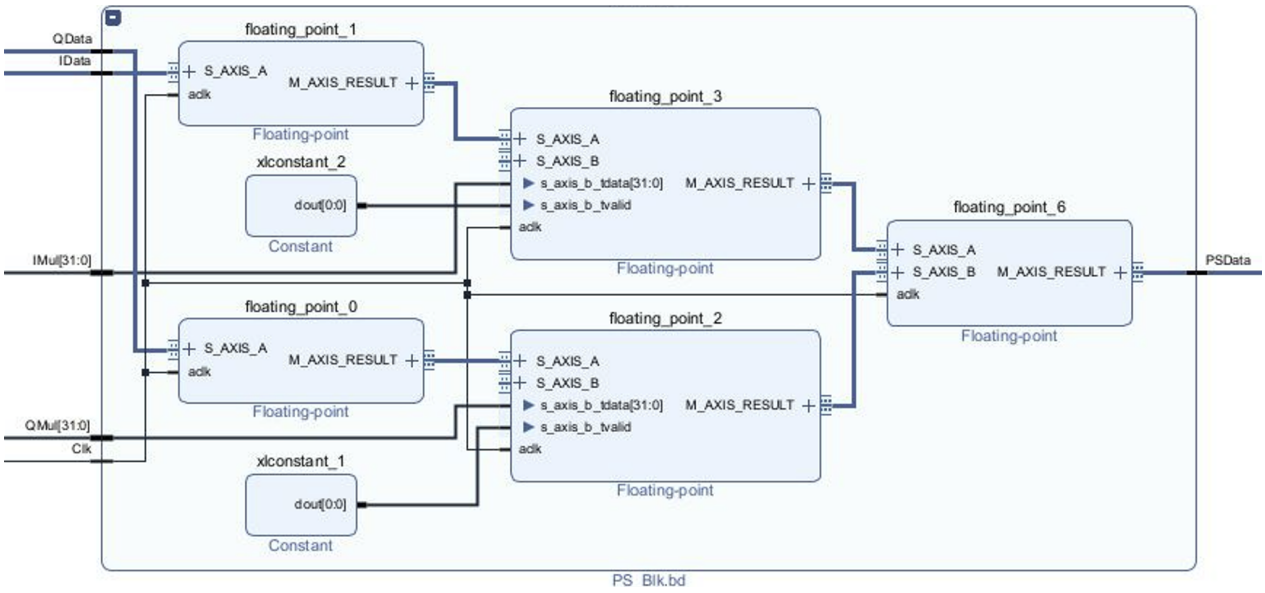


Figure 6: Internal block diagram of PS_Blk for digital phase shifting

5.1.2 RF Data Converter Configuration

The Zynq Ultrascale+ RFSoC supports high-speed data conversion via its integrated RF Data Converter IP. As part of this project, both DAC and ADC tiles were configured to support real-time beamforming operations.

DAC Configuration (Tile 228)

- **DAC Tile 228** was enabled and configured with a sampling rate of 4 GSPS.
- **Interpolation Mode** was set to 20x for DUC0 and bypassed (Off) for DUC1.
- **Analog Output Format** was real for both channels.
- **Mixer Mode** for DUC0: Real $\rightarrow$ Real with Coarse mixer at 0 Hz.

ADC Configuration (Tiles 226 and 227)

- **ADC Tiles 226 and 227** were configured for a sampling rate of 4 GSPS.
- Each tile used a reference clock of 125 MHz with internal PLLs enabled.
- The **NCO frequency was set to 1.65 GHz**, enabling digital downconversion of L-band RF signals to baseband.
- Each tile had a fabric clock of 200 MHz and a clock out frequency of 31.25 MHz.
- The configuration allowed for direct RF sampling of high-frequency inputs and digital processing using the FPGA fabric.

These ADCs captured real-time signals for digital processing, while the DAC generated the phase-shifted outputs. Multi-tile sync was not used since only a few tiles were active and phase alignment was handled in the digital domain. The below figures give the RFDC configuration.

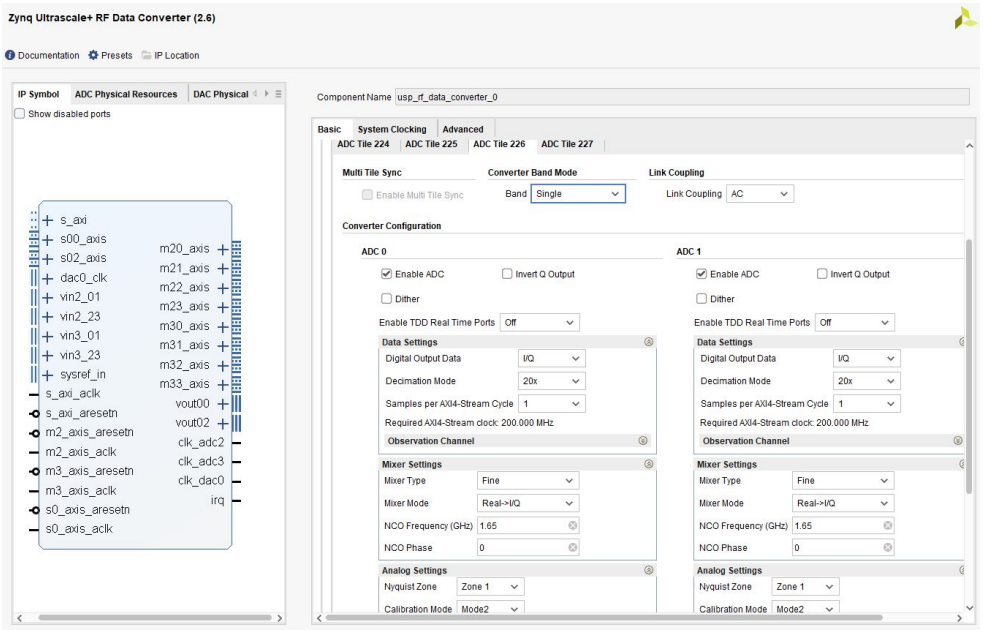


Figure 7: ADC Tile 226

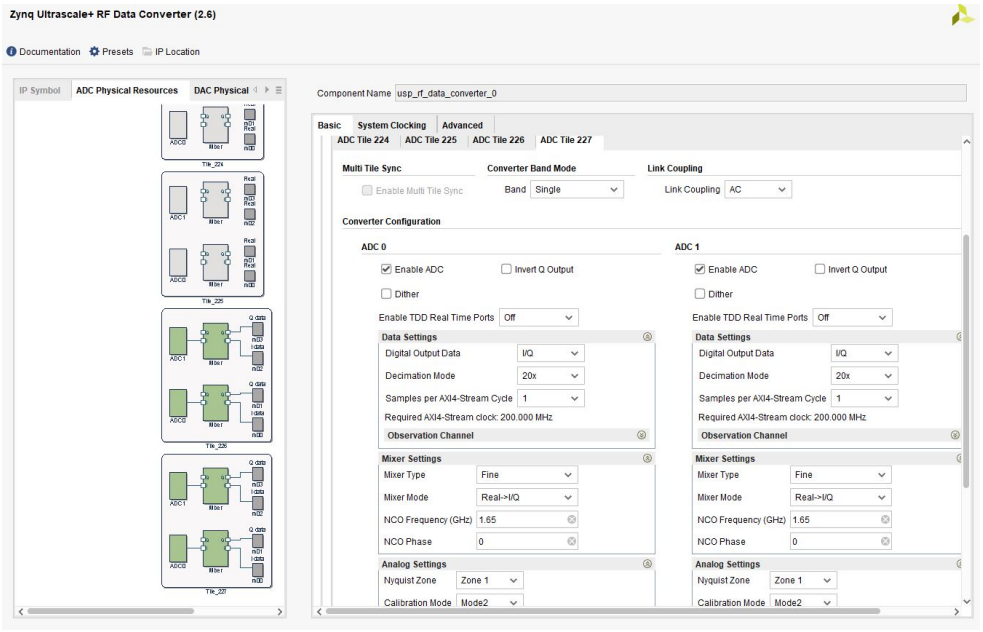


Figure 8: ADC Tile 227

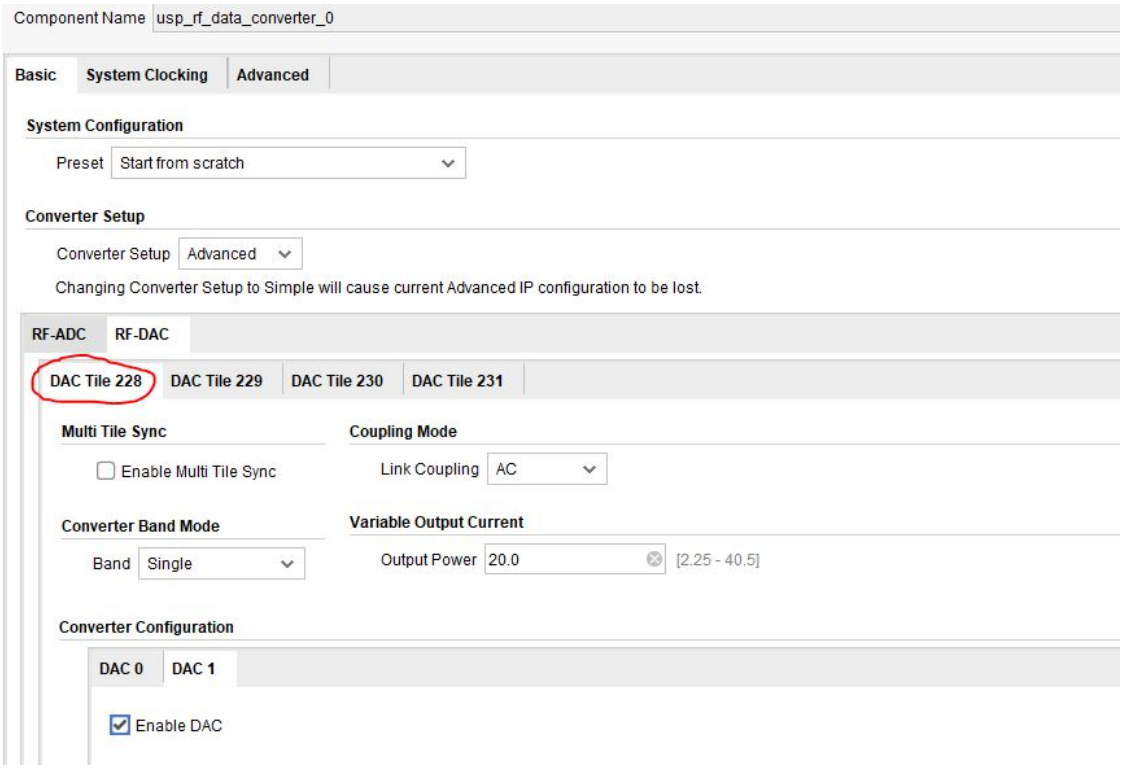


Figure 9: DAC Tile 228 enabled

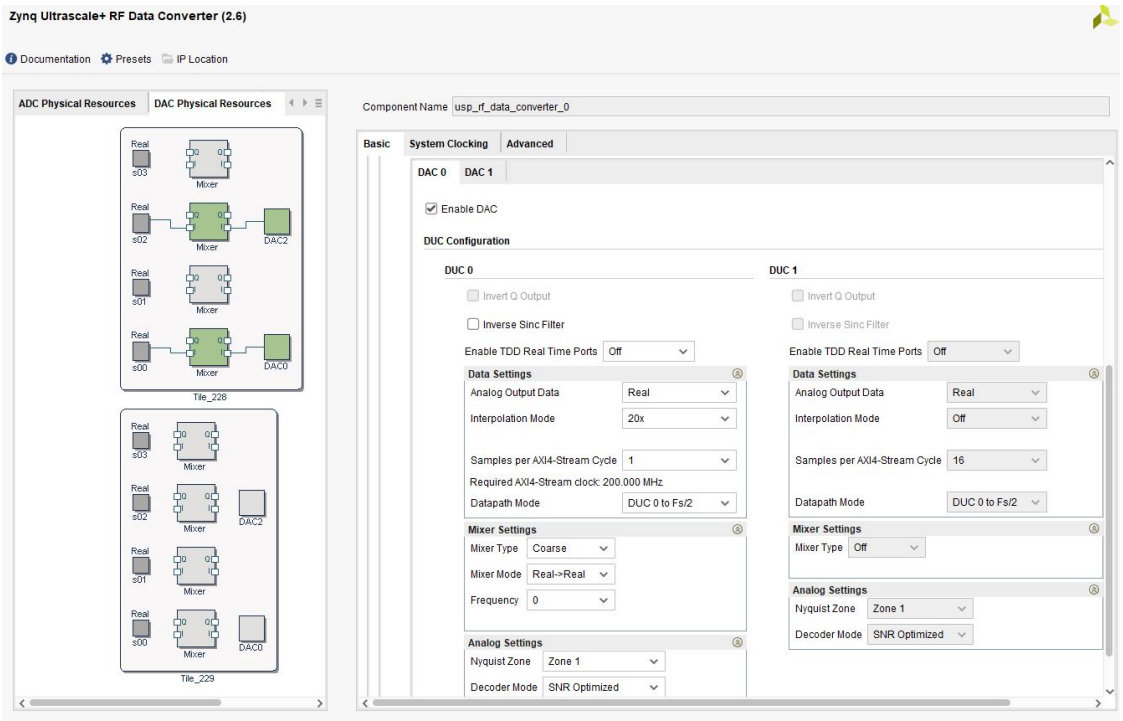


Figure 10: DUC0 and DUC1 Configuration for DAC Tile 228

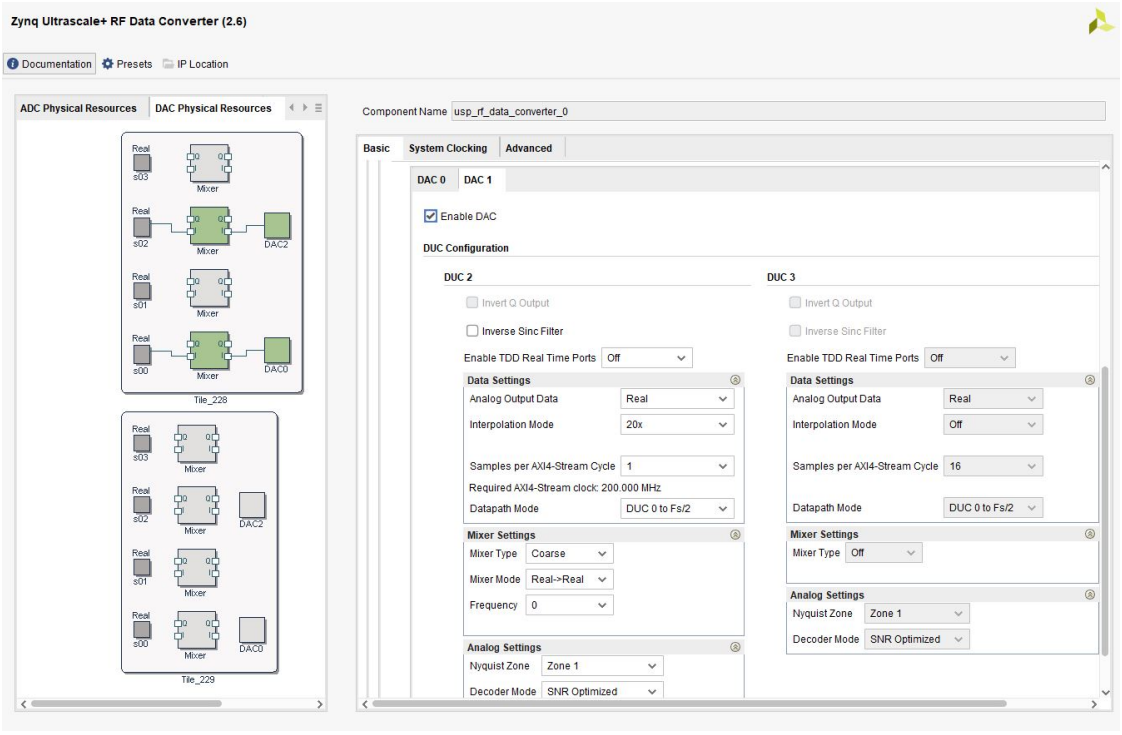


Figure 11: DUC2 and DUC3 Configuration for DAC Tile 228

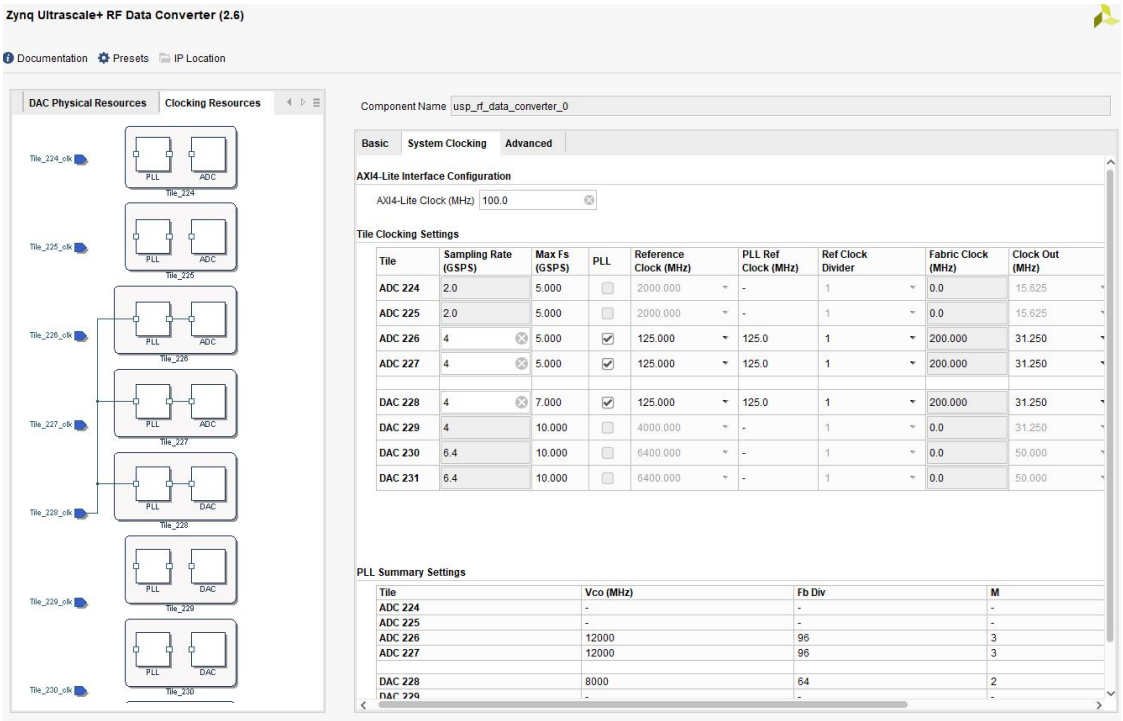


Figure 12: Clocking configuration across ADC and DAC tiles

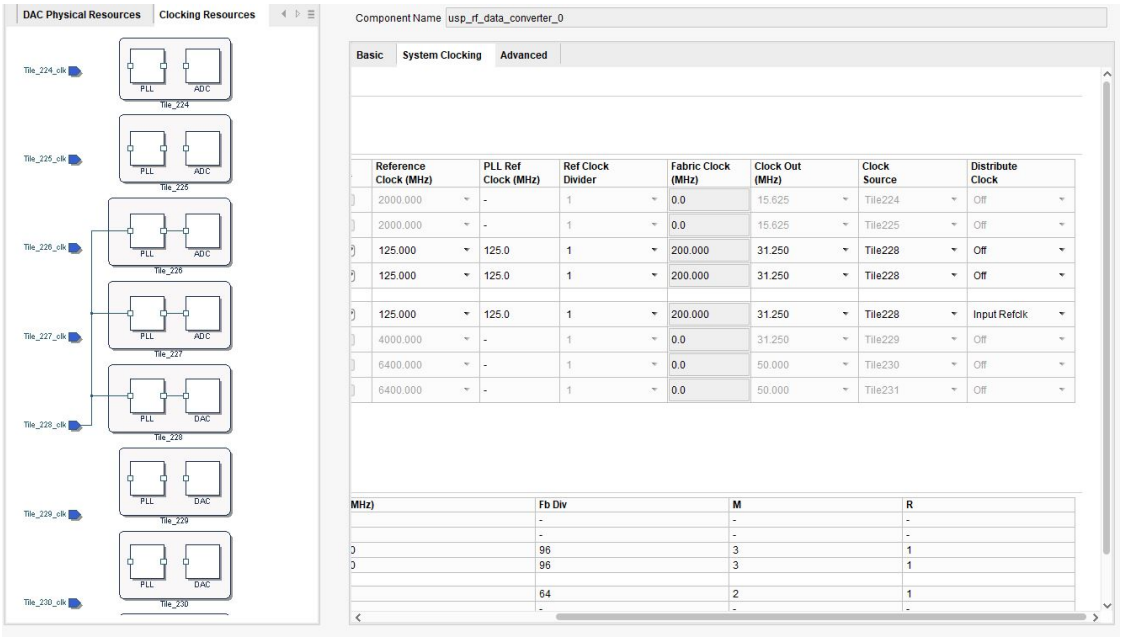


Figure 13: Clocking configuration continued

5.1.3 Other block Configurations

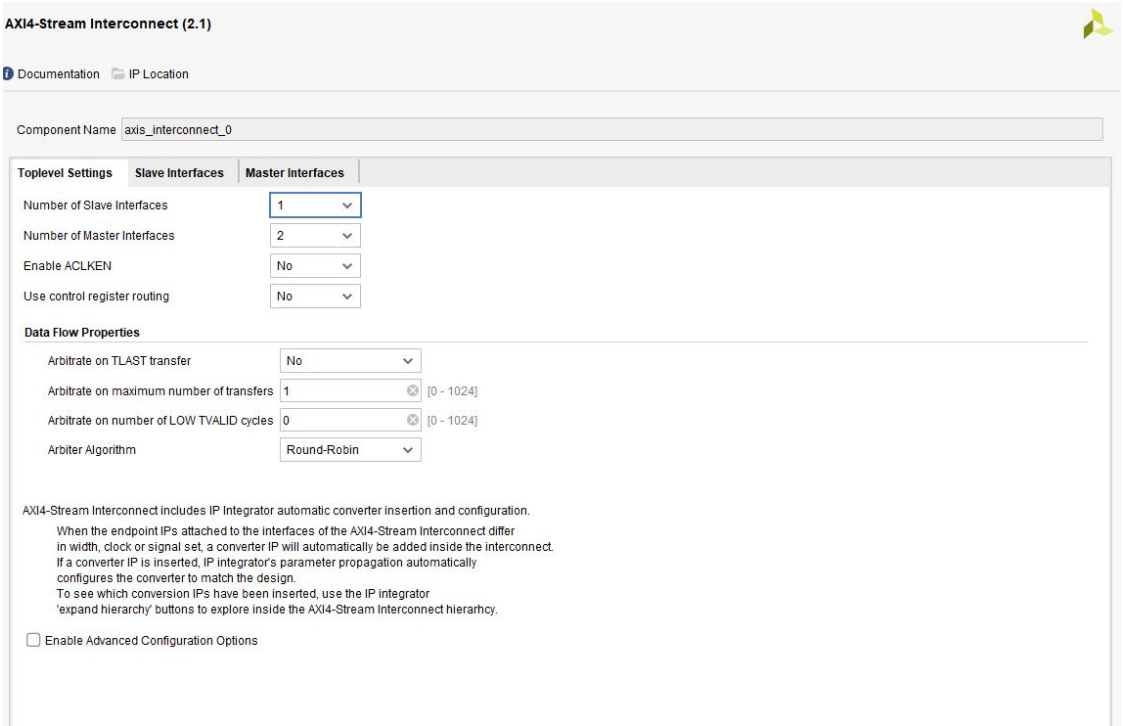


Figure 14: AXI4-Stream Interconnect

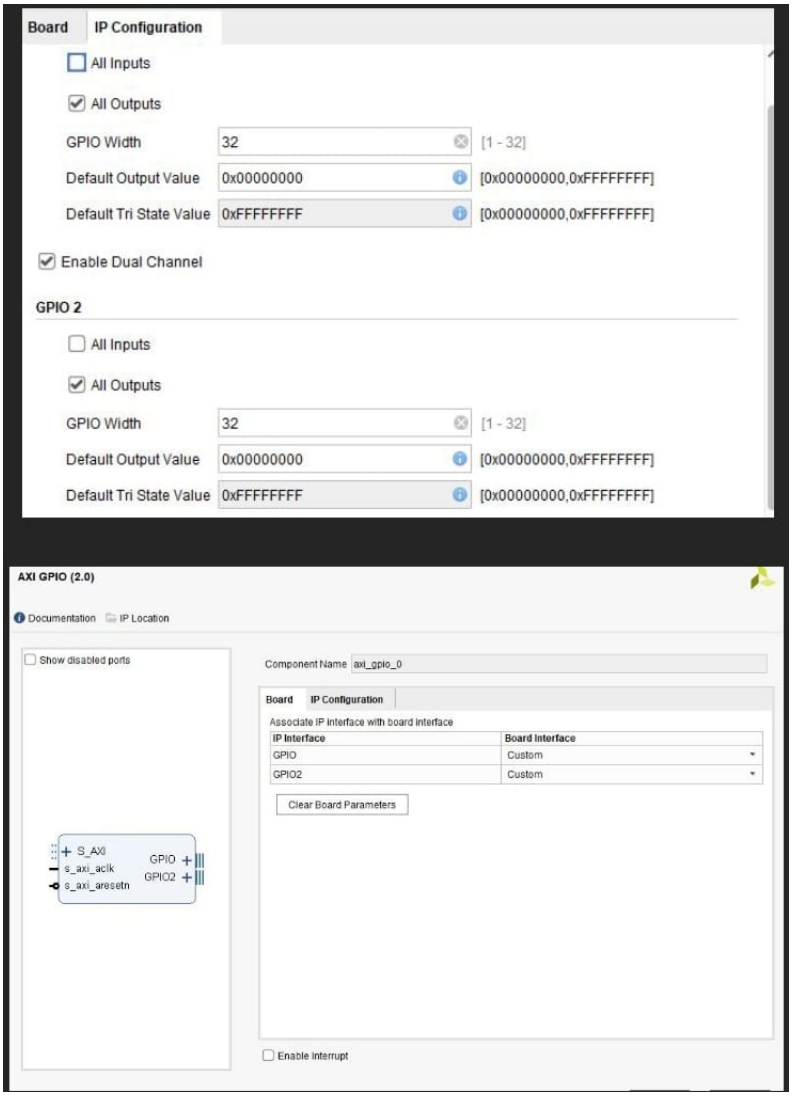


Figure 15: AXI GPIO

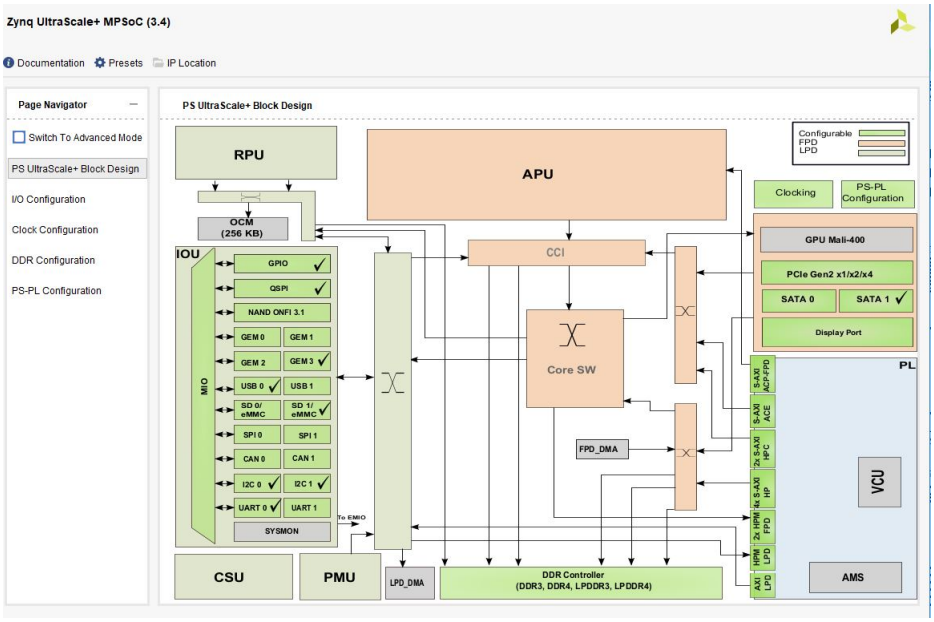


Figure 16: Zynq Ultrascale+ MPSoC

5.2 Beam Steering using Orbitron Data

Real-time beam steering is achieved by tracking the satellite position using Orbitron 3.71. Orbitron provides live updates of azimuth and elevation angles for a selected satellite, based on the observer's ground coordinates.

These angles are transmitted to the ZCU208 board via UART in the form of a formatted string:

`WAAA,EEED`

where `AAA` and `EEE` represent the azimuth and elevation in degrees, respectively.

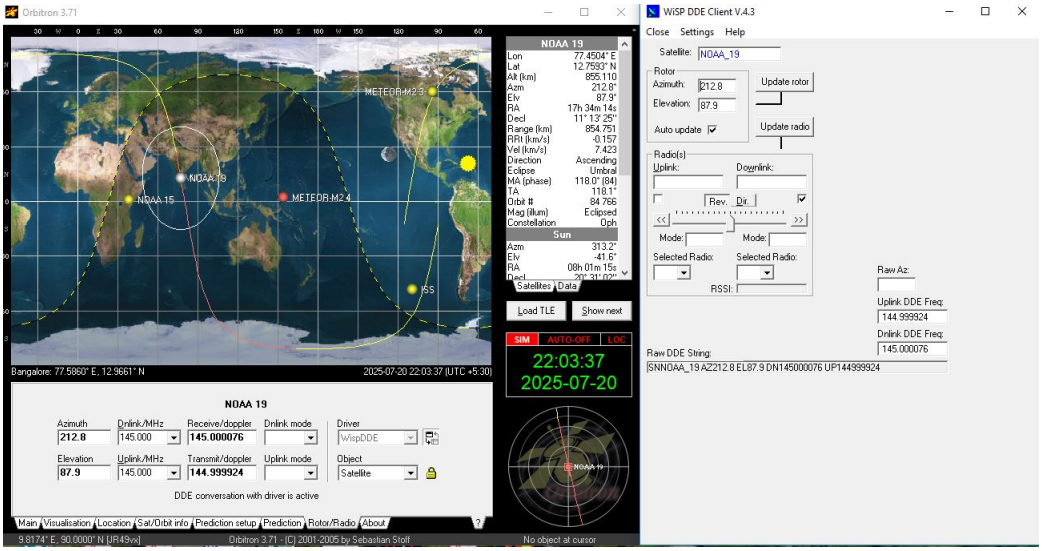


Figure 17: Orbitron and Wisp DDE

On receiving the data, the system computes the required phase shift for each antenna element using the following relation:

$$\phi_n = \frac{2\pi d_n \sin(\theta)}{\lambda}$$

where:

- ϕ_n is the phase shift for the n^{th} element,
- d_n is the distance of the n^{th} element from the reference,
- θ is the desired beam direction (derived from elevation),
- λ is the wavelength, computed from the RF center frequency ($f_c = 1.72$ GHz).

To account for hardware-induced delays and imbalances, a one-time *phase calibration* was performed at 1.72 GHz by comparing the in-phase and phase-shifted signals on an oscilloscope.

The calibration angles were then sent to the ZCU208 via UART using a dedicated format: BXXX

where `B` is the antenna element number (0 to 3), and `XXX` is the phase offset in degrees. The observed offsets were stored internally and added to the computed phase shifts during beam steering. The data was transmitted using a serial terminal such as PuTTY and applied dynamically to ensure that all antenna elements were phase-aligned prior to transmission.

The computed and calibrated phase shifts are then converted into amplitude control signals using:

$$I_n = A \cdot \cos(\phi_n), \qquad Q_n = A \cdot \sin(\phi_n)$$

These I and Q values are sent to digital multipliers inside the FPGA to apply the desired phase alignment across the antenna array, enabling dynamic beam steering toward the satellite in real time.

5.2.1 C Programming in Vitis

The Vitis toolchain was utilized to implement high-level control logic (in C), running on the ARM processing system of the ZCU208. The main tasks handled by the software were:

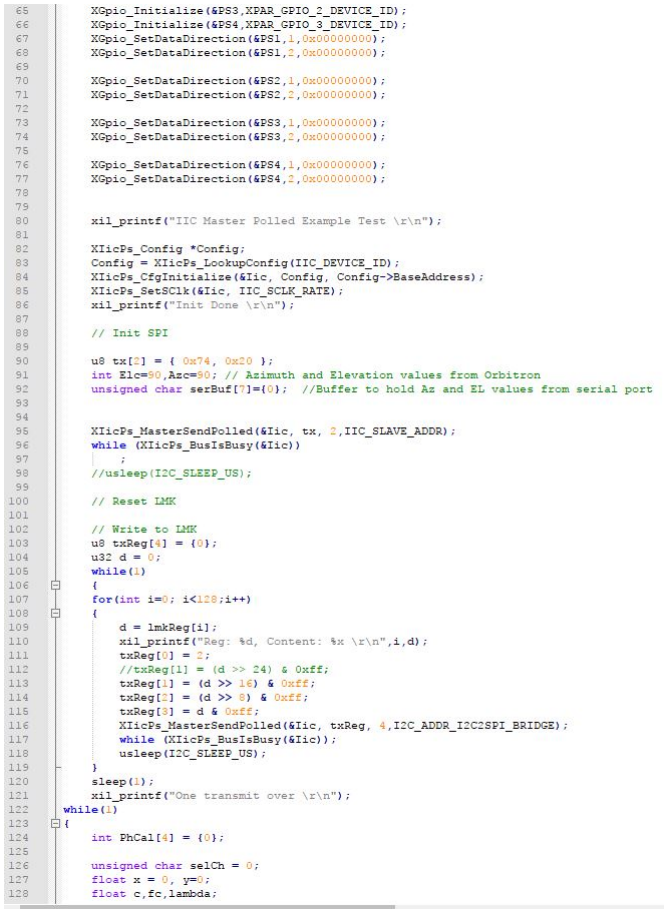
- Writing to LMK registers for configuring clock frequencies and synchronization on the ZCU208.
- Passing real-time phase values to the VHDL phase shifter block, allowing dynamic beam steering.
- Implementing user-controlled interfaces (via UART) to update phase values through external software such as Orbitron.

```c
/***************************** Include Files *********************************/
#include "xparameters.h"
#include "xiicps.h"
#include "xil_printf.h"
#include "sleep.h"
#include <stdio.h>
#include "platform.h"
#include "xil_printf.h"
#include "xgpio.h"
#include "xuartps.h"
#include <math.h>
/************************** Constant Definitions *****************************/

/*
 * The following constants map to the XPAR parameters created in the
 * xparameters.h file. They are defined here such that a user can easily
 * change all the needed parameters in one place.
 */
#define IIC_DEVICE_ID       XPAR_XIICPS_1_DEVICE_ID
#define I2C_ADDR_I2C2SPI_BRIDGE 0x2F /* I2C2SPI bridge i2c address */

// The slave address to send to and receive from.
#define IIC_SLAVE_ADDR      0x74
#define IIC_SCLK_RATE       100000
#define I2C_SLEEP_US 1000U /* I2C sleep period */

// The following constant controls the length of the buffers to be sent and received with the IIC.

#define TEST_BUFFER_SIZE    10

/**************************** Type Definitions *******************************/


/***************************** Function Prototypes ***************************/

/************************** Variable Definitions *****************************/
XUartPs COM1;
XIicPs Iic;     /**< Instance of the IIC Device */
//ADC and DAC 125MHz
const u32 lmkReg[] = {0x000090, 0x000010, 0x000200, 0x000306, 0x0004D0, 0x00055B, 0x000600, 0x000C51, 0x000D04, 0x01006C, 0x010155, 0x010255, 0x010301, 0x01042Z, 0x010500, 0x01067B, 0x010703, 0x01086C, 0x0
//125 MHz
/*
 * The following buffers are used in this example to send and receive data
 * with the IIC.
 */
u8 SendBuffer[TEST_BUFFER_SIZE] = {0x74, 0x05, 0x12,0x13,0x34,0x56,0x78,0x34,0x56,0x78};    /**< Buffer for Transmitting Data */
u8 RecvBuffer[TEST_BUFFER_SIZE] = {0};    /**< Buffer for Receiving Data */

/*****************************************************************************/
/**
* Main function to call the polled master example.
* @return   XST_SUCCESS if successful, XST_FAILURE if unsuccessful.
* @note     None.
******************************************************************************/
int main(void)
{
    unsigned char phaseVal[3];
    int angle=0;
    XUartPs_CfgInitialize(&COM1,XUartPs_LookupConfig(XPAR_PSU_UART_0_DEVICE_ID), XPAR_PSU_UART_0_BASEADDR);
    XUartPs_SetBaudRate(&COM1, 9600);

    XGpio PS1,PS2,PS3,PS4;
    XGpio_Initialize(&PS1,XPAR_GPIO_0_DEVICE_ID);
    XGpio_Initialize(&PS2,XPAR_GPIO_1_DEVICE_ID);
```

Figure 18: Initialization of I²C communication and LMK register array in Vitis

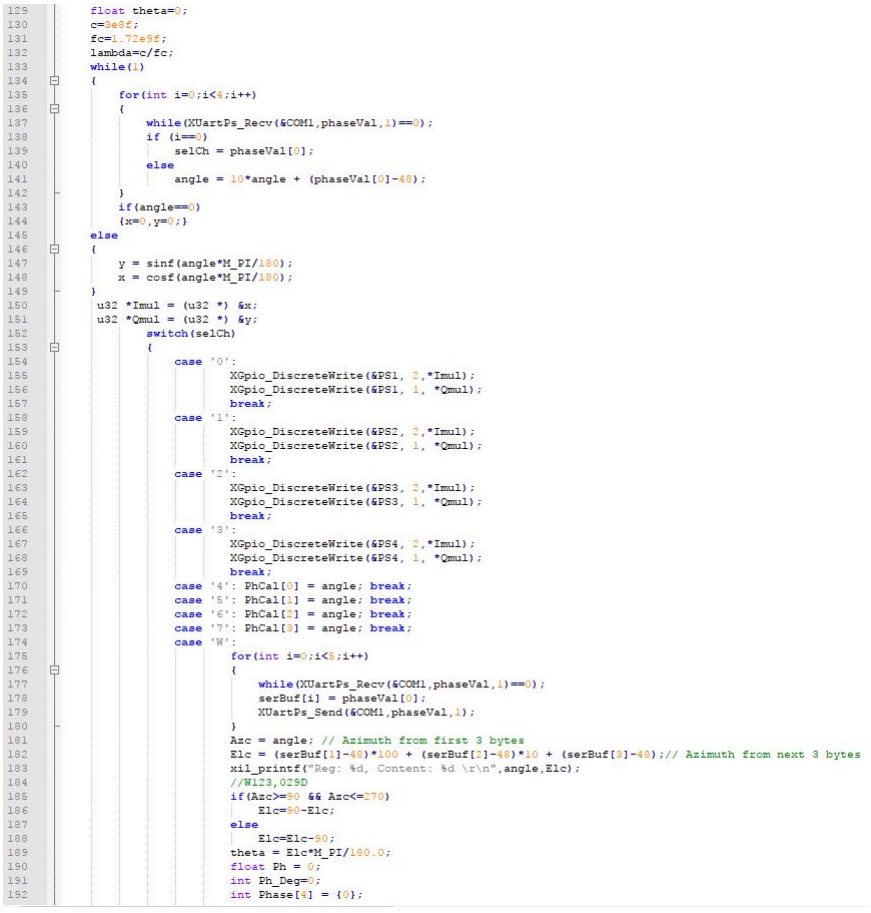
The LMK register values were generated using Texas Instruments' TICS Pro software. These values were embedded within the source code and written to the LMK device via the I²C interface during initialization. Simultaneously, AXI GPIO interfaces were configured to enable direction and data control for the four-element antenna array.

Real-time satellite tracking was achieved by communicating with Orbitron software via UART. Azimuth and elevation values from Orbitron were parsed and dynamically translated into phase shift commands for the VHDL-based phase shifter logic. These values were calculated according to array geometry and were employed to electronically steer the beam corresponding to satellite movement. To ensure precise alignment at the operating frequency of 1.72 GHz, element-specific phase calibration values derived experimentally through oscilloscope measurements were implemented. Control flow, clock programming, and beam steering calculations were accomplished using C on the ARM core of the ZCU208 through the Vitis toolchain.



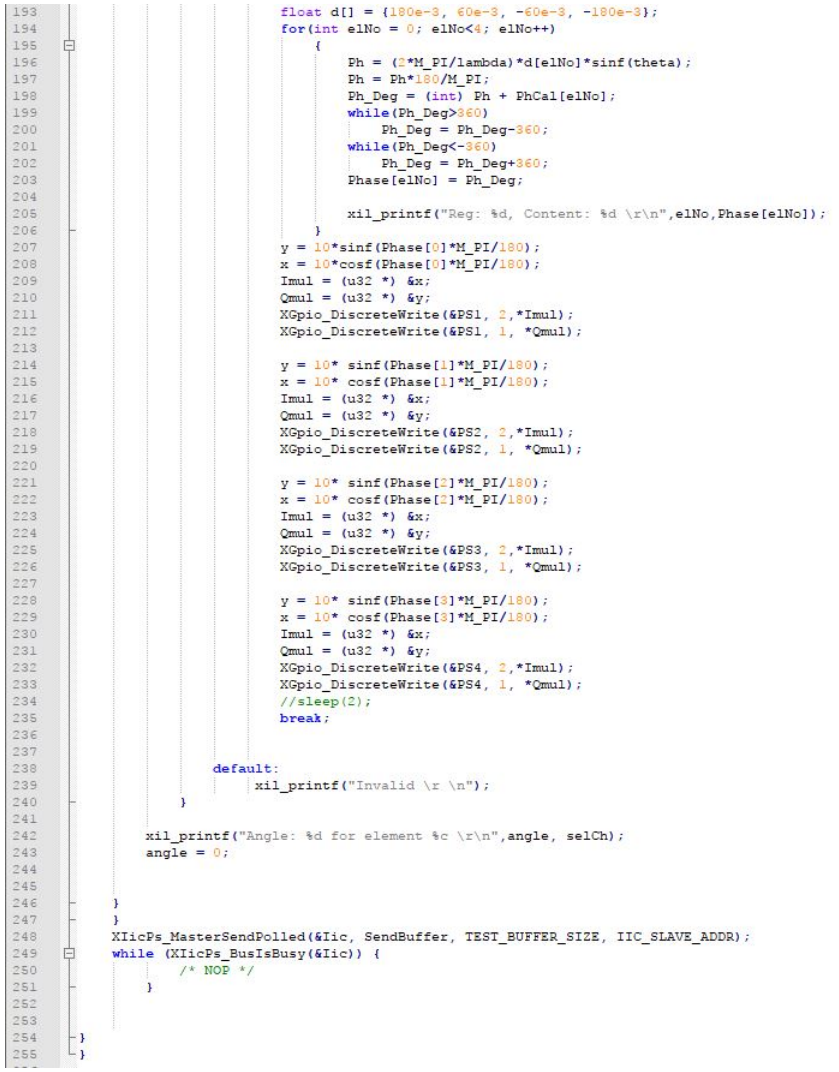
```
    XGpio_Initialize(&PS3,XPAR_GPIO_2_DEVICE_ID);
    XGpio_Initialize(&PS4,XPAR_GPIO_3_DEVICE_ID);
    XGpio_SetDataDirection(&PS1,1,0x00000000);
    XGpio_SetDataDirection(&PS1,2,0x00000000);

    XGpio_SetDataDirection(&PS2,1,0x00000000);
    XGpio_SetDataDirection(&PS2,2,0x00000000);

    XGpio_SetDataDirection(&PS3,1,0x00000000);
    XGpio_SetDataDirection(&PS3,2,0x00000000);

    XGpio_SetDataDirection(&PS4,1,0x00000000);
    XGpio_SetDataDirection(&PS4,2,0x00000000);


    xil_printf("IIC Master Polled Example Test \r\n");

    XIicPs_Config *Config;
    Config = XIicPs_LookupConfig(IIC_DEVICE_ID);
    XIicPs_CfgInitialize(&Iic, Config, Config->BaseAddress);
    XIicPs_SetSClk(&Iic, IIC_SCLK_RATE);
    xil_printf("Init Done \r\n");

    // Init SPI

    u8 tx[2] = { 0x74, 0x20 };
    int Elc=90,Azc=90; // Azimuth and Elevation values from Orbitron
    unsigned char serBuf[7]={0};  //Buffer to hold Az and EL values from serial port


    XIicPs_MasterSendPolled(&Iic, tx, 2,IIC_SLAVE_ADDR);
    while (XIicPs_BusIsBusy(&Iic))
        ;
    //usleep(I2C_SLEEP_US);

    // Reset LMK

    // Write to LMK
    u8 txReg[4] = {0};
    u32 d = 0;
    while(1)
    {
    for(int i=0; i<128;i++)
    {
        d = lmkReg[i];
        xil_printf("Reg: %d, Content: %x \r\n",i,d);
        txReg[0] = 2;
        //txReg[1] = (d >> 24) & 0xff;
        txReg[1] = (d >> 16) & 0xff;
        txReg[2] = (d >> 8) & 0xff;
        txReg[3] = d & 0xff;
        XIicPs_MasterSendPolled(&Iic, txReg, 4,I2C_ADDR_I2C2SPI_BRIDGE);
        while (XIicPs_BusIsBusy(&Iic));
        usleep(I2C_SLEEP_US);
    }
    sleep(1);
    xil_printf("One transmit over \r\n");
while(1)
{
    int PhCal[4] = {0};

    unsigned char selCh = 0;
    float x = 0, y=0;
    float c,fc,lambda;
```

Figure 19: Writing LMK register values and configuring UART interface



```
float theta=0;
c=3e8f;
fc=1.72e9f;
lambda=c/fc;
while(1)
{
    for(int i=0;i<4;i++)
    {
        while(XUartPs_Recv(&COM1,phaseVal,1)==0);
        if (i==0)
            selCh = phaseVal[0];
        else
            angle = 10*angle + (phaseVal[0]-48);
    }
    if(angle==0)
    {x=0,y=0;}
else
{
    y = sinf(angle*M_PI/180);
    x = cosf(angle*M_PI/180);
}
 u32 *Imul = (u32 *) &x;
 u32 *Qmul = (u32 *) &y;
        switch(selCh)
        {
            case '0':
                    XGpio_DiscreteWrite(&PS1, 2,*Imul);
                    XGpio_DiscreteWrite(&PS1, 1, *Qmul);
                    break;
            case '1':
                    XGpio_DiscreteWrite(&PS2, 2,*Imul);
                    XGpio_DiscreteWrite(&PS2, 1, *Qmul);
                    break;
            case '2':
                    XGpio_DiscreteWrite(&PS3, 2,*Imul);
                    XGpio_DiscreteWrite(&PS3, 1, *Qmul);
                    break;
            case '3':
                    XGpio_DiscreteWrite(&PS4, 2,*Imul);
                    XGpio_DiscreteWrite(&PS4, 1, *Qmul);
                    break;
            case '4': PhCal[0] = angle; break;
            case '5': PhCal[1] = angle; break;
            case '6': PhCal[2] = angle; break;
            case '7': PhCal[3] = angle; break;
            case 'W':
                    for(int i=0;i<5;i++)
                    {
                        while(XUartPs_Recv(&COM1,phaseVal,1)==0);
                        serBuf[i] = phaseVal[0];
                        XUartPs_Send(&COM1,phaseVal,1);
                    }
                    Azc = angle; // Azimuth from first 3 bytes
                    Elc = (serBuf[1]-48)*100 + (serBuf[2]-48)*10 + (serBuf[3]-48);// Azimuth from next 3 bytes
                    xil_printf("Reg: %d, Content: %d \r\n",angle,Elc);
                    //W123,029D
                    if(Azc>=90 && Azc<=270)
                        Elc=90-Elc;
                    else
                        Elc=Elc-90;
                    theta = Elc*M_PI/180.0;
                    float Ph = 0;
                    int Ph_Deg=0;
                    int Phase[4] = {0};
```

Figure 20: Phase Calibration data handling and elevation correction



```
                    float d[] = {180e-3, 60e-3, -60e-3, -180e-3};
                    for(int elNo = 0; elNo<4; elNo++)
                        {
                            Ph = (2*M_PI/lambda)*d[elNo]*sinf(theta);
                            Ph = Ph*180/M_PI;
                            Ph_Deg = (int) Ph + PhCal[elNo];
                            while(Ph_Deg>360)
                                Ph_Deg = Ph_Deg-360;
                            while(Ph_Deg<-360)
                                Ph_Deg = Ph_Deg+360;
                            Phase[elNo] = Ph_Deg;

                            xil_printf("Reg: %d, Content: %d \r\n",elNo,Phase[elNo]);
                        }
                    y = 10*sinf(Phase[0]*M_PI/180);
                    x = 10*cosf(Phase[0]*M_PI/180);
                    Imul = (u32 *) &x;
                    Qmul = (u32 *) &y;
                    XGpio_DiscreteWrite(&PS1, 2,*Imul);
                    XGpio_DiscreteWrite(&PS1, 1, *Qmul);

                    y = 10* sinf(Phase[1]*M_PI/180);
                    x = 10* cosf(Phase[1]*M_PI/180);
                    Imul = (u32 *) &x;
                    Qmul = (u32 *) &y;
                    XGpio_DiscreteWrite(&PS2, 2,*Imul);
                    XGpio_DiscreteWrite(&PS2, 1, *Qmul);

                    y = 10* sinf(Phase[2]*M_PI/180);
                    x = 10* cosf(Phase[2]*M_PI/180);
                    Imul = (u32 *) &x;
                    Qmul = (u32 *) &y;
                    XGpio_DiscreteWrite(&PS3, 2,*Imul);
                    XGpio_DiscreteWrite(&PS3, 1, *Qmul);

                    y = 10* sinf(Phase[3]*M_PI/180);
                    x = 10* cosf(Phase[3]*M_PI/180);
                    Imul = (u32 *) &x;
                    Qmul = (u32 *) &y;
                    XGpio_DiscreteWrite(&PS4, 2,*Imul);
                    XGpio_DiscreteWrite(&PS4, 1, *Qmul);
                    //sleep(2);
                    break;


                default:
                    xil_printf("Invalid \r \n");
            }

        xil_printf("Angle: %d for element %c \r\n",angle, selCh);
        angle = 0;


    }
    }
    XIicPs_MasterSendPolled(&Iic, SendBuffer, TEST_BUFFER_SIZE, IIC_SLAVE_ADDR);
    while (XIicPs_BusIsBusy(&Iic)) {
            /* NOP */
        }


}
}
```

Figure 21: Final phase output to antenna elements via GPIO

Every phase value calculated, after wrapping and calibrating to the 0–360° range, is transformed into sine and cosine components. These are scaled and quantized to write to AXI GPIO interfaces. This provides synchronized phase shifts among elements so the antenna array can steer the beam precisely based on real-time satellite position data.

This hardware–software co-development offered real-time reconfigurability of beam steering angles and close coupling between control and signal generation layers.

6 Limitations and Future Work

Although the implementation achieved its desired objectives, certain challenges were noted:

- Manual phase calibration introduced human error and restricted repeatability.
- Phase calibration varied across operating frequencies.
- Synchronization across RFSoC tiles was not always consistent.

Future enhancements can overcome these challenges and enhance system capabilities:

- Calibration automation through feedback algorithms.
- Refining tile synchronization using advanced clocking schemes.
- Scaling the system to larger arrays and faster interfaces.

7 Conclusion

The project represented a major milestone in bringing RFSoC technology to the field of practical digital beamforming. Beyond developing the end-to-end signal chain, it required understanding the nuances of clock synchronization, data converter configuration, and embedded system control. The combination of Orbitron-based satellite tracking with real-time phase shifting pointed towards the dynamic nature of the system.

Working at the intersection of hardware description (VHDL), embedded programming (Vitis), and RF system design offered a unique and challenging learning experience. The limitations faced revealed important design constraints and possibilities for improvement. Overall, the internship encouraged technical development and problem solving skills necessary for advanced work in satellite communication and signal processing systems.

8 Appendices

8.1 List of Abbreviations

- **RFSoC** – Radio Frequency System on Chip
- **ADC** – Analog to Digital Converter
- **DAC** – Digital to Analog Converter
- **GPIO** – General Purpose Input Output
- **I^2C** – Inter-Integrated Circuit
- **UART** – Universal Asynchronous Receiver Transmitter
- **NCO** – Numerically Controlled Oscillator
- **AXI** – Advanced eXtensible Interface
- **DBF** – Digital Beamforming
- **LRPT** – Low Resolution Picture Transmission
- **HRPT** – High Resolution Picture Transmission

8.2 List of Figures

List of Figures

8.3 List of Tables

List of Tables