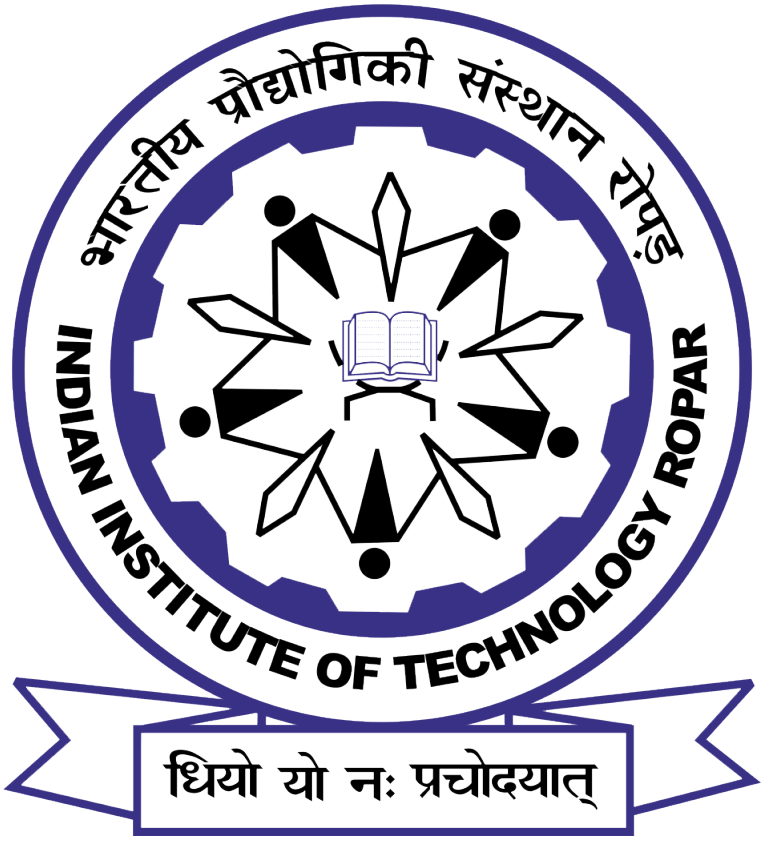


Indian Institute of Technology Ropar

Department of Electrical Engineering

Aman Bhagat 2022eeb1152, Nakul 2022eeb1192

Supervisor: Dr. Brijesh kumbani

May 12, 2025

Contents

List of Figures

Chapter 1

Introduction

1.1 Introduction to project

In this project, we worked with the RFSoC 4x2 development board, a powerful platform built around the AMD-Xilinx Zynq UltraScale+ RFSoC (XCZU48DR). This board integrates high-speed ADCs and DACs, a quad-core ARM Cortex-A53 processor, a dual-core Cortex-R5F, and extensive programmable logic, making it ideal for high-performance signal processing applications such as software-defined radio (SDR).

As part of the development process, we began by cloning a GitHub repository, provided through the official PYNQ reference documentation, which contained the necessary overlays and support files tailored for the RFSoC 4x2 platform. This repository served as our base environment for working with PYNQ and interacting with the hardware through Python.

In the second stage of the project, we focused on the hardware design aspect by attempting to generate a custom bitstream using Vivado, the FPGA development platform from AMD-Xilinx. This involved working with the board's programmable logic resources and integrating required IP blocks to customize the hardware configuration.

This report details our workflow, implementation steps, and findings from using the RFSoC 4x2 board in a real-world signal processing scenario. A reference link to the PYNQ documentation and the GitHub repository used in this project is provided at the end of the report.

1.2 Background

Premature infants—those born before 37 weeks of gestation—often suffer from underdeveloped organs and compromised immune systems, making continuous monitoring of vital signs such as breathing rate, heart rate, and oxygen saturation crucial, especially in Neonatal Intensive Care Units (NICUs). Traditionally, this monitoring relies on wired skin-contact sensors, which, although essential, can cause discomfort, restricted movement, skin irritation, and in severe cases, pressure-induced skin damage.

To address these limitations, this project explores a completely contactless radar-based breathing monitoring system. The proposed solution utilizes a continuous wave (CW) radar setup to non-invasively track breathing patterns, aiming to be deployed in real NICU environments. Radar technology offers several advantages: it is low-cost, low-power, operates through blankets and incubator walls, and is unaffected by lighting conditions or skin pigmentation.

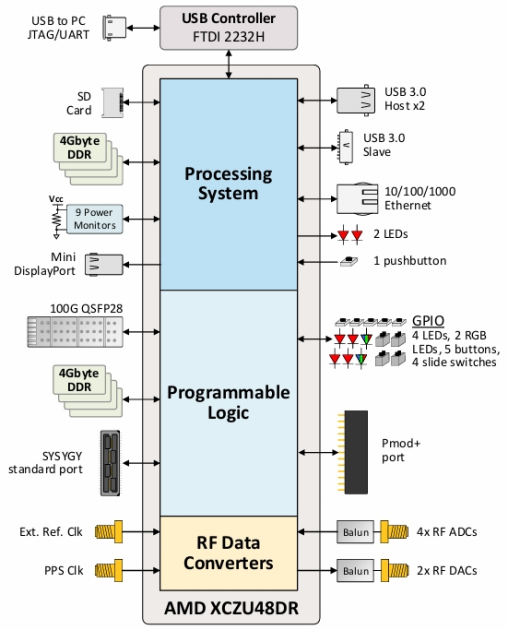
Given that approximately 15 million babies are born prematurely each year worldwide, and many face life-threatening respiratory conditions such as apnea, bradycardia, and bron-

chopulmonary dysplasia, there is a growing need for safe, non-contact monitoring solutions. While existing alternatives like photoplethysmography, dynamic light scattering, and video-based analysis show promise, they often fall short under variable lighting, limited visibility, or skin-tone-related constraints. Radar-based methods overcome these challenges, offering robust performance even under less-than-ideal clinical conditions.

This project aims to demonstrate the feasibility and effectiveness of such a radar-based solution using the RFSoC 4x2 board, combining hardware acceleration with real-time signal processing and intelligent data analysis.

Chapter 2

Hardware Information



In this project, we employed the RFSoC 4x2 development board as the core hardware platform to implement and analyze high-speed signal processing systems. The RFSoC 4x2 is a compact yet powerful system designed by RealDigital, built around the AMD-Xilinx Zynq UltraScale+ Gen3 RFSoC (XCZU48DR).
This platform integrates high-speed ADCs and DACs, programmable FPGA fabric, and an ARM-based processing system. It supports signal sampling at rates up to 5GSPS and waveform generation up to 9.85GSPS.
Additionally, it offers high-speed interfaces (USB 3.0, Gigabit Ethernet, QSFP28) and PYNQ-based programming, enabling intuitive high-level control. This board served as the foundation for our experimental design.

Figure 2.1: Architecture and key components of the RFSoC 4x2 development board.

2.1 PYNQ (Python Productivity for Zynq) Framework

PYNQ (Python Productivity for Zynq) is an open-source framework that allows users to control the RFSoC 4x2 hardware using Python. Built on Jupyter Lab, it provides an easy-to-use interface for programming and interacting with the board without needing traditional hardware design languages. With PYNQ, users can manage data converters, configure hardware overlays, and analyze signals in real-time, making complex signal processing tasks more accessible and efficient.

Here is the LaTeX code to include the technical description of the **RFSoC 4x2** board in your report, formatted with a section and an itemized list for readability:

2.2 RFSoC 4x2 Features

The **RFSoC 4x2** board is a feature-rich platform built around the AMD-Xilinx Zynq UltraScale+ RFSoC (XCZU48DR), making it ideal for high-speed digital signal processing and software-defined radio (SDR) applications. Key features include:

- 4x 14-bit RF ADCs with sampling rates up to 5 GSPS
- 2x 14-bit RF DACs with output rates up to 9.85 GSPS
- Dual 4 GB DDR4 memory banks (one for PS and one for PL)
- Quad-core ARM Cortex-A53 and dual-core Cortex-R5F processors
- 100G QSFP28 port, Gigabit Ethernet, USB 3.0, and DisplayPort
- Integrated programmable logic with 930K logic cells and 4.2K DSP slices
- Precision clocking system with external sync and PPS input
- Support for SYZYGY and Pmod+ expansion connectors
- Full compatibility with PYNQ, Vitis, and Vivado development tools

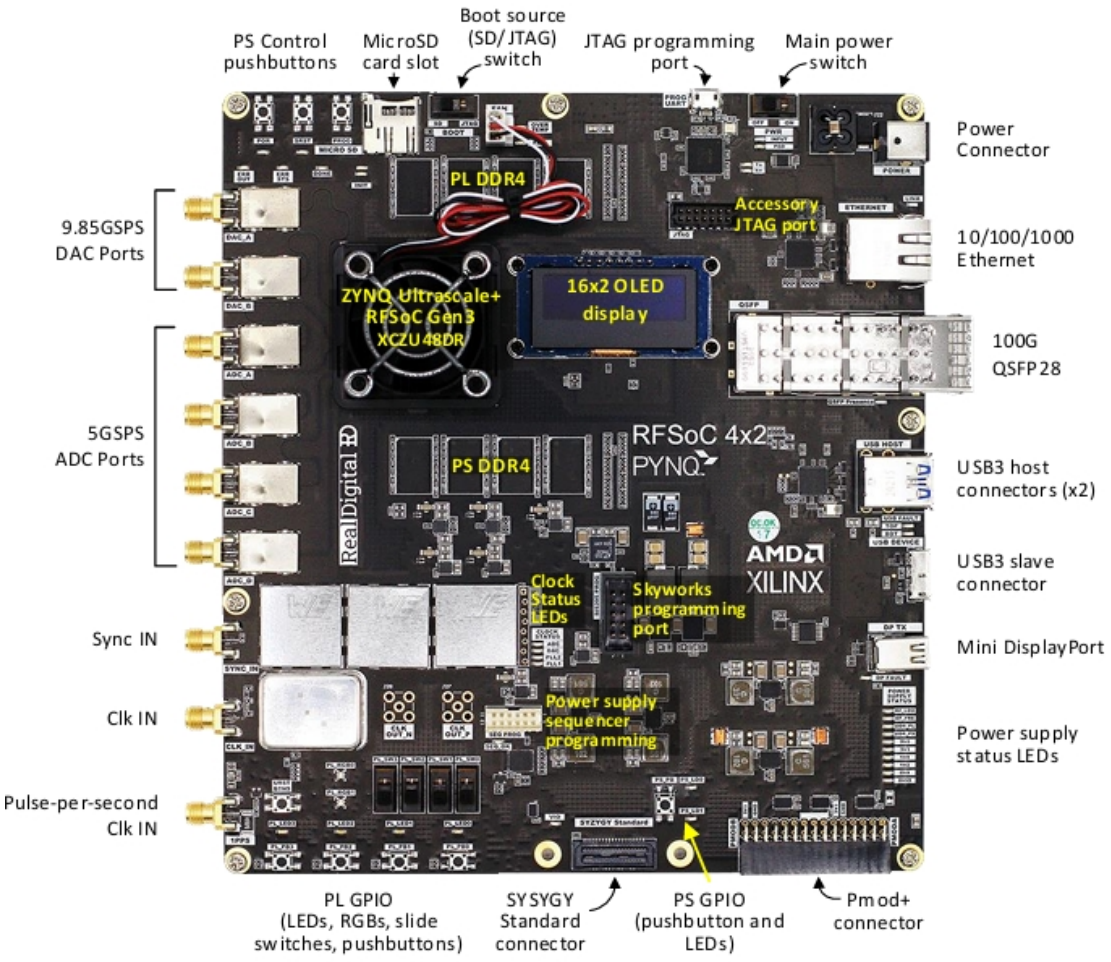


Figure 2.2: RFSoC 4*2 development boardl

2.3 Memory

- It features two separate 4GB DDR4 memory modules, each 64-bit wide and operating at 2400MHz.
- One memory bank is connected to the Processing System (PS) for general computation.
- The other is linked to the Programmable Logic (PL), optimized for handling large volumes of streaming data from the ADCs and DACs.
- Additionally, the Zynq RFSoC includes on-chip memory such as 256KB SRAM in the PS and 60Mb of UltraRAM and block RAM in the PL.

2.4 Data Ports

- The board includes a 100G QSFP28 port for high-speed data offload, suitable for Ethernet or fiber connections.
- It supports Gigabit Ethernet with time stamping and jumbo frame support.
- USB interfaces include two USB 3.0 host ports and one USB 3.0 slave port, as well as a USB 2.0-based JTAG/UART port for programming and debugging.
- Additional interfaces include a microSD card slot, DisplayPort, and expansion connectors like SYZYGY and Pmod+ for custom I/O.

Chapter 3

Getting Started with the Board

3.1 Follow the Steps

Setting up the RFSoC 4x2 board is straightforward and requires just a few steps:

- Insert the MicroSD card preloaded with the PYNQ image into the SD slot.
- Set the boot mode to SD using the boot switch located near the card slot.
- Connect the USB 3.0 and power cables, then turn on the board using the main power switch.
- Wait for initialization—status LEDs will turn on, and the OLED display will show the board's IP address after boot-up.
- Access the board via a web browser at `http://192.168.3.1/lab` and log in with the password `xilinx`.

Once logged in, you'll be in the Jupyter Lab environment, ready to run Python notebooks, control hardware, and begin experimenting with the powerful features of the RFSoC 4x2.

3.2 Step by step Guidance of Board setup with figures:

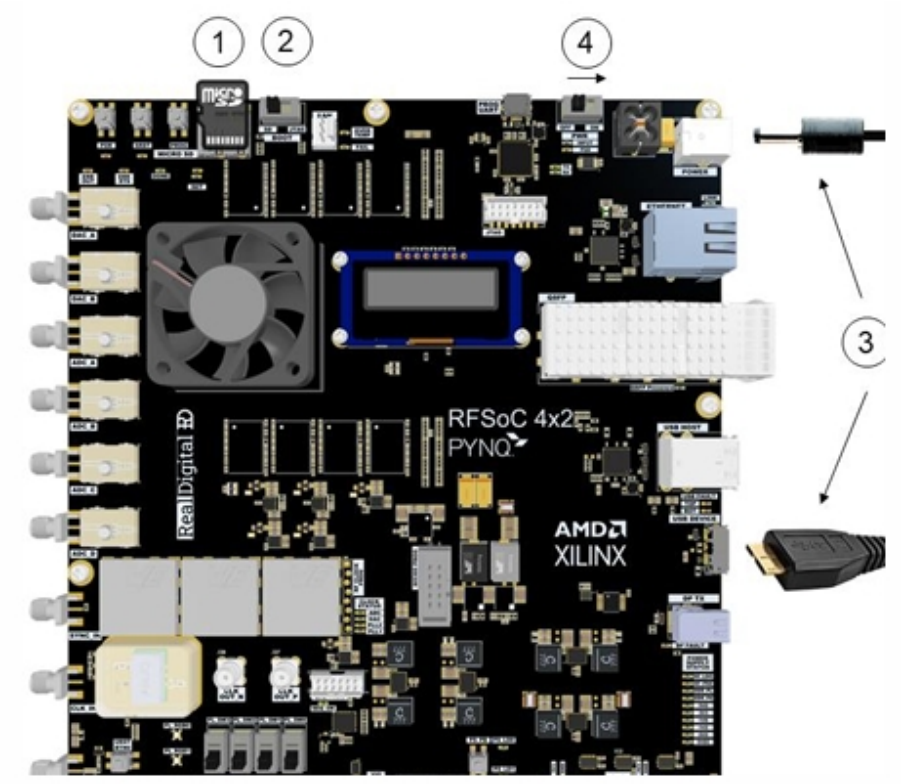


Figure 3.1: Insert the SD card and set up the board as shown in fig.

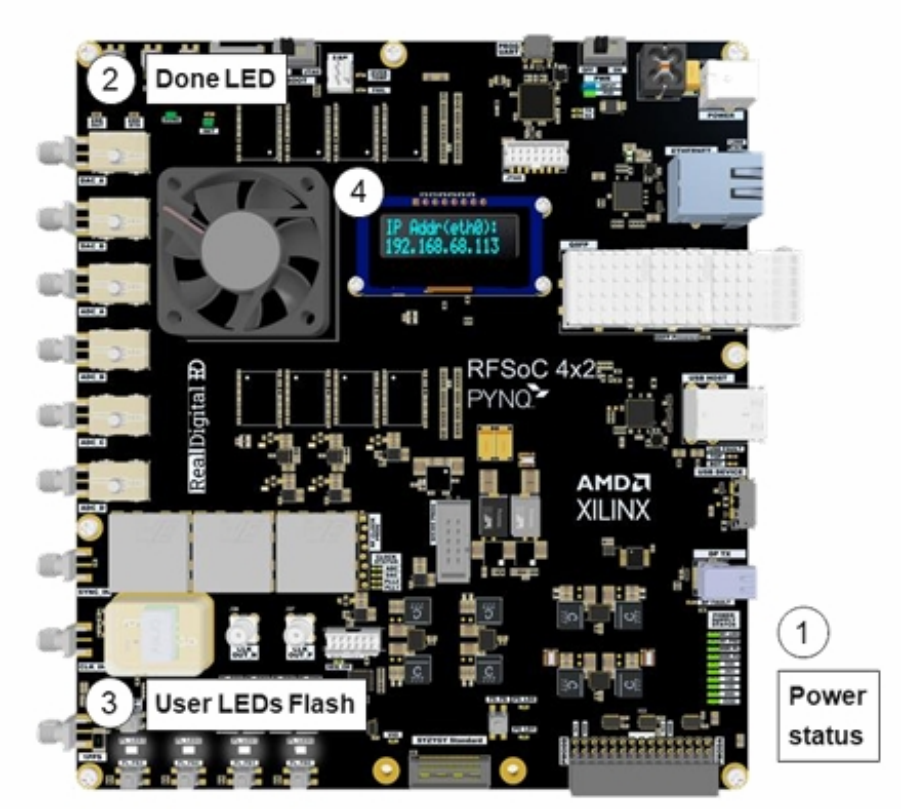


Figure 3.2: Switch ON power supply ensure power status LED turn on and IP address should visible on Display.

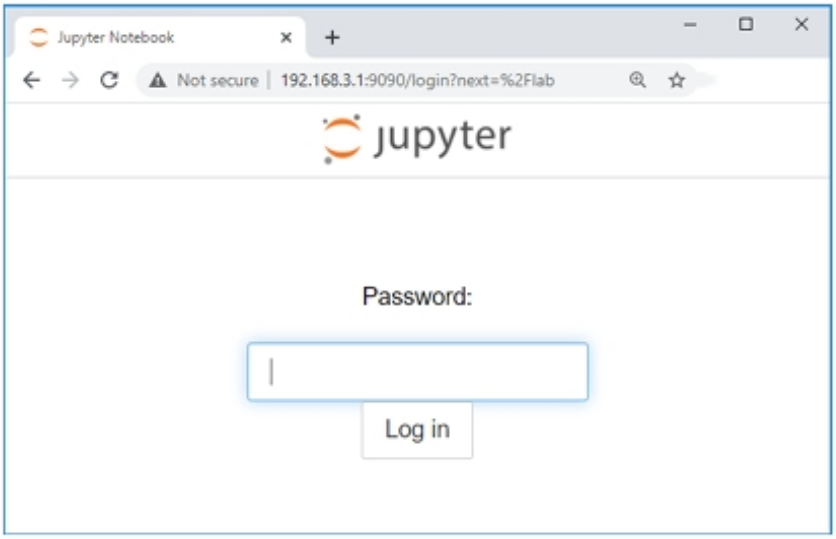


Figure 3.3: Enter xilinx as password

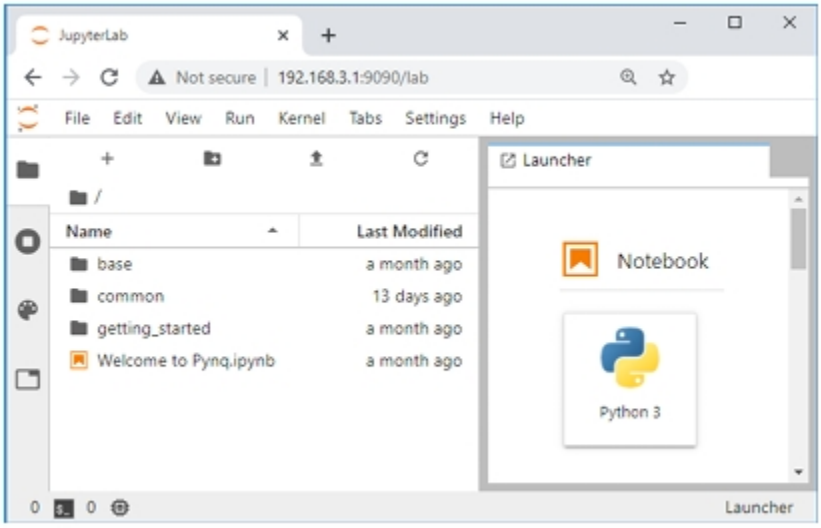


Figure 3.4: Start Exploring

Chapter 4

Methodology

This project was carried out following a structured sequence of steps, each clearly outlined to provide step-by-step guidance throughout the process.

4.1 BPSK / QPSK Transmitter Diagram Analysis

- System Generator : The interface between Simulink and Xilinx FPGA tools. It translates high-level Simulink models into VHDL/Verilog for implementation on Xilinx FPGAs.
- DMA Read : Handles data transfer from memory to the DUT (Design Under Test). DMA allows high-throughput transfer of streaming data used for modulation.
- DUT (Design Under Test) : - This is the core logic block where the BPSK/QPSK modulation happens.
- Convert Signal Converts internal signal format (e.g., fixed-point) to an output-compatible form (e.g., analog or floating-point).
- Control Switches and Constants (1/0 blocks) Set test/control conditions like modulation scheme (1 for QPSK, 0 for BPSK), enable flags, etc.

4.2 BPSK / QPSK Receiver Diagram Analysis

1. **QPSK System**
   This block implements Quadrature Phase Shift Keying (QPSK) modulation.
   Converts input data into complex QPSK symbols, represented by their real and imaginary parts.
2. **Valid Signal Generator**
   A constant signal source generating a logical high (1).
   Acts as a "Valid" indicator for the AXI interface, signaling that the data is valid.
3. **Complex AXI-Stream Slave Interface**
   Receives and processes streaming complex data using the AXI handshaking protocol.
4. **SSR Halfband Lowpass Decimator**
   Performs two main operations:
   – *Lowpass Filtering:* Removes high-frequency components.

– *Decimation:* Reduces sampling rate while preventing aliasing.
The SSR (Super Sample Rate) filter is hardware-efficient due to reduced multiplier requirements.

5. **CIC Decimator**
   Further reduces the sampling rate using a multiplier-free Cascaded Integrator-Comb (CIC) structure.
   Ideal for FPGA designs where large decimation factors are needed.
6. **Coarse Frequency Synchronisation**
   Corrects large frequency offsets between transmitter and receiver.
   Brings signal frequency closer to reference before fine tuning.
7. **RRC Receive Filter**
   Implements a Root Raised Cosine (RRC) filter to minimize intersymbol interference (ISI).
   Matches transmitter pulse shaping to ensure optimal detection and reduce noise.
8. **Time Synchronisation**
   Aligns sampling points with symbol centers.
   Compensates for timing offsets, ensuring accurate symbol recovery.
9. **Carrier Synchronisation Block**
   Performs carrier recovery and phase synchronization.
   Aligns the receiver's local oscillator with the incoming carrier frequency.
10. **Frame Synchronisation Block**
    Detects frame boundaries and extracts timing data.
    Helps identify start and end of frames and tracks modulation symbol counts.
11. **AXI-Stream Master Interface**
    Converts synchronized data into AXI-Stream format.
    Prepares the output for further digital signal processing.
12. **Observation Point Block**
    Serves as a debug interface to monitor internal system signals.
    Aggregates data for real-time observation without disturbing data flow.
13. **Reset Subsystem Block**
    Manages all reset and control signals across the system.
    Supports global and targeted resets (time, phase, frame).
    Also distributes modulation mode to other system blocks.

4.3 Vivado Guidance

To Generate a Bitstream for the RFSoC 4x2 Board Using Vivado, Follow These Steps:

4.3.1 Creating a New Project in Vivado (RFSoC 4x2 Board)

Follow the steps below to create a new project in Vivado and set it up for the RFSoC 4x2 board:

1. **Launch Vivado**
   Open the Vivado software.
   The main interface includes three sections:

- Quick Start
- Tasks
- Learning Center

2. **Start a New Project**
   Under the Quick Start section, click "*Create New Project*" (use "*Open Project*" if you already have one).
   A popup window titled "Create a New Vivado Project" will appear. Click *Next*.
3. **Set Project Name and Location**
   Enter your project name.
   Choose the location where the project will be saved.
   Click *Next* to continue.
4. **Select Project Type**
   Choose "*RTL Project*" (Register Transfer Level).
   Tick the checkbox "*Do not specify sources at this time*" (optional, if you haven't added files yet).
   Click *Next*.
5. **Select Default Part or Board**
   In the Default Part window, switch to the *Boards* tab.
   Search for and select "*RFSoC 4x2*" (make sure the board files are installed).
   Click *Next*, then *Finish*.
6. **Project Setup Complete**
   You are now in the Vivado project workspace, where you can add design sources, IP blocks, and start building your hardware design.

4.3.2 Creating Block Design and Generating Bitstream in Vivado

Follow the steps below to create a block design and generate a bitstream for the RFSoC 4x2 board using Vivado:

1. **Create Block Design (if using IP Integrator)**
   - Go to *Flow Navigator* → *Create Block Design*.
   - Add essential IPs like Zynq RFSoC, clocks, AXI interfaces, etc.
   - Configure the IP blocks according to your design.
   - Connect the blocks appropriately.
   - Click *Validate Design* to ensure there are no errors.
2. **Generate Output Products**
   After validating, right-click the block design and select *Generate Output Products*.
3. **Create HDL Wrapper**
   Right-click the block design and choose *Create HDL Wrapper* → *Let Vivado manage wrapper*.
4. **Run Synthesis**
   From the *Flow Navigator*, click *Run Synthesis*.
   Wait for the synthesis process to complete.
   Review and click *OK* to proceed.

5. **Run Implementation**
   After synthesis, click *Run Implementation*.
   Wait for placement and routing to complete.
   Review timing and utilization reports.
6. **Generate Bitstream**
   After successful implementation, click *Generate Bitstream*.
   Wait for the bitstream (`.bit`) file to be created.
7. **Export Hardware**
   If using Vitis or PYNQ, go to *File* → *Export Hardware* → Choose with or without bitstream.

Chapter 5

Results and Future Plan

5.1 Result

In this project, we successfully followed the study material provided by PYNQ and cloned the recommended GitHub repository as part of the setup process. Using the Vivado platform, we attempted to generate a bitstream for the RFSoC 4x2 board by building the required hardware blocks and configurations. While we completed the design steps, the bitstream generation process encountered an error due to a bit mismatch issue, which prevented successful compilation. This error highlights the importance of ensuring compatibility between IP configurations and board specifications, which we plan to address in future iterations.

5.2 Future Plan

This section outlines the key steps taken to validate the system design, address bitstream generation issues, and optimize the radar-based breathing monitoring solution.

- **Debug Bitstream Generation Issue**
  Identify and correct the bit mismatch error encountered during bitstream generation in Vivado.
- **Verify IP Configuration Compatibility**
  Ensure all IP blocks are correctly configured and compatible with the RFSoC 4x2 board specifications.
- **Successfully Generate Bitstream**
  Complete synthesis, implementation, and bitstream generation without errors.
- **Deploy Radar-Based Breathing System**
  Test the contactless radar-based system in a controlled, simulated NICU environment.
- **Integrate Machine Learning Models**
  Apply and evaluate models (e.g., Random Forest) for accurate breathing rate estimation.
- **Improve System Robustness**
  Enhance signal processing to handle motion artifacts and external interference.

Chapter 6

Reference

- https://www.rfsoc-pynq.io/tutorial.html
- https://www.rfsoc-pynq.io/overlays.html
- https://www.rfsoc-pynq.io/index.html
- https://github.com/strath-sdr/rfsoc$_r adio$