Priya’s Interview Essentials: Core Technical Points

General Quick-Recall

- **Verilog, RTL, and FPGA:**
  - *Always mention*: You’re hands-on with Verilog for RTL design, digital logic, testbenches, simulation, and implementation on FPGA (mainly Basys 3, sometimes Zedboard).
- **Tools Used:**
  - **Vivado** for FPGA design, simulation & debugging
  - **MATLAB** for signal test vectors & validation
  - **Arduino IDE** for embedded microcontroller projects
  - **STM32Cube/Keil** for STM32 MCU work

Project Quick Reference

1. DRDO - FPGA Digital Demodulator

- **Hardware:**
  - *FPGA* (for all logic, parallel real-time processing)
  - **ADC7980** (dual, SPI, 200kHz sampling, 5V)
- **Key Design Choices:**
  - Used **FPGA** (not microcontroller/DSP) for parallelism, high throughput, and low latency in radar signal processing.
  - **CORDIC** used on FPGA for real-time sine/cosine generation (for I/Q demodulation), very resource-efficient (no multipliers, just shifts & adds).
  - **DDS** (Direct Digital Synthesizer): Implemented using CORDIC for reference wave generation.
  - Full flow: ADC → FPGA (CORDIC) → Multiplier → LPF → I/Q Output
- **Protocols:**
  - **SPI** for ADC-FPGA interfacing (signals: CS, CLK, DIN, DOUT)
- **Results:**
  - Achieved **97% detection accuracy**, verified with MATLAB test vectors
  - Debugged angle precision/bit-width issues by increasing resolution
- **Testing:**
  - Modular testbenches, MATLAB vs Verilog outputs, real signal + noise testing

2. Secure Voting Machine (FPGA)

- **Hardware:**
  - *FPGA (Basys 3)*, 7-segment display, switches, LEDs
- **Design:**
  - All logic (voting, result display, security) in Verilog
  - **Finite State Machine (FSM):**
    - **IDLE:** Wait for correct password
    - **VOTING_OPEN:** Accept button votes, count per candidate
    - **VOTING_CLOSED:** Show winner, result display
  - **Password-protection:** Only allows voting if correct password entered via switches
  - **Tamper-resistance:** Votes stored only in hardware counters (not modifiable externally)
- **Tech Details:**
  - **Debouncer module**: Eliminates multiple votes due to button bounce
  - **Clock divider:** Slows 100MHz input for human-readable displays (1Hz for state transitions, 2Hz for blinking)
  - **Binary-to-BCD conversion**: For correct vote display on 7-segment
- **Scalability:**
  - Add candidates by adding more buttons/counters
- **Possible Enhancements:**

- Biometric/keypad auth, LCD/touch display, IoT/remote monitoring

---

3. CO GasShield (Embedded, Arduino)

- **Hardware:**
  - **Arduino Nano (ATmega328P, 5V)**
  - **MQ-7 CO sensor** (analog)
  - **SIM800L GSM module** (SMS alerts, 3V)
  - **NEO-6M GPS** (location)
  - **OLED display, Buzzer, LEDs**
- **Power Management:**
  - **Buck-boost converter:** To supply different voltages (Arduino & sensor: 5V, GSM: 3V)
- **Design:**
  - **Analog CO levels read by Arduino ADC**
  - **Multi-level alert system:**
    - *Green* (safe), *Yellow* (warning), *Red* (danger), *Critical* (SMS alert)
  - **Software filtering:** To avoid false positives from sensor noise
  - **Serial/UART** used for GSM, GPS; careful management to prevent comm interference
- **Testing/Accuracy:**
  - Achieved **98% detection accuracy** (tested with real/simulated leaks)
  - *Key learning*: User survey showed need for cost-effective solutions (target <₹1500)
- **Expansion ideas:**
  - Automatic mitigation (open windows), cloud connectivity

---

4. ICU Noise Monitor (STM32 MCU)

- **Hardware:**
  - **STM32 microcontroller** (with built-in ADC, RTC)
  - **MAX4466 microphone** (analog, to ADC)
  - **WS2812B RGB LEDs** (for visual alerts)
- **Design:**
  - **STM32's RTC**: Differentiates day (40 dBA) vs night (35 dBA) thresholds automatically
  - **LEDs:**
    - *Green* (safe), *Yellow* (warning), *Red* (alert)
  - **Timers/DMA:** Used for precise PWM to drive LEDs
  - **Debugging:** USART used to stream debug data to PC
- **Key challenge:**
  - Accurate dBA conversion from analog, solved by careful calibration and formula implementation
- **Skills Used:**
  - Embedded C, peripheral setup (ADC, RTC, PWM), real-time processing

---

5. UART Controller (Maven Silicon, FPGA)

- **Hardware:**
  - *FPGA implementation* (Verilog)
- **Design:**
  - **FSM for Transmitter:** IDLE → SENDING → DONE
  - **FSM for Receiver:** IDLE → START → DATA → STOP
  - Manages 8N1 frame (start, 8 data bits, stop), with precise timing (baud rate divider)
  - **Verification:** Testbenches checked that sent = received data
- **Key learning:**
  - Sequential logic, timing, state machine design

---

6. 4-bit ALU (NIELIT, Verilog)

- **Hardware:**
  - RTL in Verilog (simulated, could be mapped to FPGA)
- **Design:**
  - 4-bit A, B inputs; operations: Add, Sub, AND, OR, XOR, NOT, pass-through
  - **Carry out** for add/sub
  - Used **case statements** for ALU_Sel
- **Learning:**
  - Foundation for digital logic, synthesis, and testbench creation

---

What Chip/Controller Where? (Quick Table)

Project	Main Chip/Controller	Notes/Why Not MCU?
Demodulator (DRDO)	FPGA, dual ADC7980 (SPI)	FPGA for parallel, real-time signal processing. DSP/MCU too slow for radar.
Voting Machine	FPGA (Basys 3)	FSM logic, hardware security.
GasShield	Arduino Nano (ATmega328P)	Easy for analog read, sensor interface, compact for embedded.
ICU Noise Monitor	STM32 microcontroller	ADC, RTC, timers, C programming for real-time data and thresholds.
UART Controller	FPGA (any)	Sequential logic, FSMs, needs accurate timing.
ALU	RTL in Verilog (for FPGA)	Logic synthesis, simulation.

Protocols, Connections, and Gotchas

- **ADC to FPGA:** Usually via **SPI**. Need CS, CLK, DIN, DOUT; sample rate must be chosen for your signal bandwidth.
- **Sensor to Microcontroller:** Analog signal → ADC pin (Arduino/STM32)
- **UART:** Serial comm between modules (MCUs, PC, sensors)
- **Power:** Always check different modules' voltage (buck-boost if needed!)

Personal/HR Quick-Recall

- **Strengths:** Problem-solving, fast learner, detail-oriented, organized
- **Weakness:** Sometimes over-perfect, now learning to balance with deadlines; learning to ask for help sooner
- **Extra:** Always relate "I designed/tested/verified myself", "compared output to MATLAB", "wrote testbenches to validate every block"

Final Practical Tips

- Always **mention why FPGA** (parallel, low-latency, real-time; not just "because").
- **CORDIC** = resource-efficient, parallel sine/cos generation (for I/Q, radar, etc).
- **FSM** = for stepwise, state-driven hardware (voting, UART, etc).
- **Debugging:** Use testbenches for RTL/FPGA, serial/PC debug for MCU.
- **User studies:** Useful for real product decisions (cost matters!).
- **Power supply:** Always plan for different module voltages and stable comms.

Introduction

Hi, I'm Priya Raj. I'm currently pursuing my M.Tech in Electronics Design and Technology at NIT Calicut, with a strong foundation in digital and embedded system design. I completed my B.Tech in ECE from JSS Noida. My academic and industry experience spans real-time hardware design, embedded systems, and digital architectures implemented on FPGAs." During my internship at Ericsson, I worked on automating real-time monitoring for telecom networks, improving system uptime and diagnostic efficiency.

On the academic front, I've designed and implemented FPGA-based systems including a secure voting machine and a synchronous demodulator for DRDO, both demonstrating high accuracy and real-time performance. These projects involved integrating digital PLLs, dual-ADC interfaces, and state machine control, which has given me hands-on exposure to RTL design, timing analysis, and protocol interfacing.

I'm particularly passionate about building efficient, scalable hardware solutions, and I see Qualcomm's innovation-driven environment as a perfect place to contribute meaningfully at the silicon and system level. Outside of work, I'm from Kanpur, UP, and I enjoy exploring the mountains and experimenting with cooking.

"By hardware design, I mean designing digital logic using Verilog and implementing it on FPGA. For example, in my DRDO project, I built a synchronous demodulator by designing RTL blocks for phase detection and signal reconstruction, interfacing dual ADCs, and validating the system using Vivado on FPGA hardware."

- "I'm doing well, thank you! Excited to be here and looking forward to our conversation."
- "I'm good, thank you! How about you?"

"FPGA-based architecture" means **the entire digital system (like demodulators, voting machines, etc.) was built and implemented on an FPGA**, where you defined how the hardware behaves using code (Verilog/SystemVerilog), not software like C.

Ericsson

Q1: What did you do at Ericsson?" or "Tell me about your Ericsson internship?"):

“At Ericsson, I worked as an Automation Engineer intern with a team focused on improving the reliability of Orange Belgium’s telecom network.
My main role was to build smart monitoring systems that could check the health of the network in real time.

I also created ,custom alerts ,dashboards and visual reports that made it easy for the team to see which parts of the network were performing well and which parts needed attention. the tools I used was OMC,BMC which helped us analyze and troubleshoot issues faster.

Another part of my work involved monitoring the performance of high-speed network bands and making sure they were running efficiently. Because of the automation and alert systems I built, the number of outages dropped by nearly 40%, and the network reliability improved to about 90%.

Question 2: "Can you elaborate on how you achieved 90% network availability for Orange Belgium?"
“When I joined, the goal was to improve network uptime. Instead of just fixing things after they broke, I helped set up a more proactive system.
I worked with senior engineers to understand which parts of the network were most critical, then created scripts and dashboards to track important metrics like latency and signal quality.
I also set alert thresholds so that we could catch issues early and take action before they became serious problems. This early-warning setup helped us keep the network running smoothly and achieve 90% uptime.”

Question 3: "How did you manage to reduce outage incidents by 40%?"**

“Some of the high-speed network bands were facing frequent outages, which really impacted service.
I was responsible for monitoring these bands in real time and jumping in quickly when something looked off.
Using performance dashboards and diagnostic tools, I tracked down issues — whether they were caused by hardware, software bugs, or wrong configurations — and helped fix them fast.
I also documented what I found, so similar problems could be solved faster next time. This hands-on, quick-response approach helped us bring down outages by 40%.”

"Can you explain how you optimized RAN system performance using BMC Helix and other diagnostic tools, and what was the impact HOW YOU IMPROVED 25% ?"**

“Yes, I used BMC Helix along with other diagnostic tools to monitor and analyze the RAN system’s performance in real time. I tracked key metrics like signal strength, latency, and throughput to identify patterns or drops in performance. Once issues were spotted, I’d dig deeper using these tools to find bottlenecks—whether from hardware, software, or network configuration. Based on the insights, I suggested and implemented optimizations like parameter tuning(**Changing some settings** in the network to make it work better.) or reconfigurations. These actions led to a 25% improvement in RAN efficiency and helped maintain stable, high-quality service for users.”

NIELIT (Trainee) - Jul 2023 - Aug 2023

Project:

Yes, during my NIELIT training, to really get hands-on with the foundational principles, I designed a **simple 4-bit Arithmetic Logic Unit (ALU)**. This involved taking basic logic operations like addition, subtraction, AND, and OR, and implementing them in Verilog. We then simulated it to make sure it performed correctly for various inputs. It was a straightforward project, but it was excellent for understanding how to translate theoretical digital logic concepts into actual hardware descriptions and verify their function, which is a core part of VLSI design.

```
module ALU_4bit (
   input  [3:0] A,
   input  [3:0] B,
   input  [2:0] ALU_Sel,    // 3-bit control signal
   output reg [3:0] Result,
   output reg CarryOut
);

always @(*) begin
   case (ALU_Sel)
      3'b000: {CarryOut, Result} = A + B;        // Addition
      3'b001: {CarryOut, Result} = A - B;        // Subtraction
      3'b010: Result = A & B;                    // AND
      3'b011: Result = A | B;                    // OR
      3'b100: Result = A ^ B;                    // XOR
      3'b101: Result = ~A;                       // NOT A
      3'b110: Result = A;                        // Pass A
      3'b111: Result = B;                        // Pass B
      default: Result = 4'b0000;
   endcase
end

Endmodule
```

3. What It Does (Features)

"The ALU takes two 4-bit inputs (`A`, `B`) and performs operations like:
Addition, Subtraction, AND, OR, XOR, NOT, pass-through based on a 3-bit control signal.
It also generates a carry-out in arithmetic operations."

🔷 5. My Role & Learning

"I designed the RTL code, created a testbench to simulate all ALU operations, and verified outputs using waveform analysis.
I learned how to write synthesizable Verilog, use conditional logic (case statements), and test digital blocks systematically."

🔷 6. Key Takeaway

"This project gave me a strong foundation in RTL design, simulation, and digital circuit behavior, which is crucial for front-end VLSI design roles."

Question 1: "Can you tell me about your experience as a trainee at NIELIT?"**

At NIELIT, my internship focused on VLSI and digital system design, which helped me build a strong practical understanding of core concepts like RTL design, simulation, and verification. I worked on hands-on projects where I applied my theoretical knowledge to real digital design problems.

During the internship, I wrote and analyzed Verilog code, created testbenches, and learned how to debug timing and logic issues. I also gained experience with the overall VLSI design flow — from writing RTL to understanding synthesis and how hardware gets optimized.

One of the most valuable parts of the internship was learning to think like a verification engineer — not just focusing on how the design works, but also how to properly test and validate it. The environment at NIELIT was very supportive and learning-driven, and I made sure to take notes, ask questions, and keep track of everything I worked on.

Overall, it gave me confidence in working with digital systems and exposed me to industry tools and workflows that are commonly used in VLSI projects."

"During my training at NIELIT, my main focus was to gain a solid understanding of VLSI Design. I successfully completed a specialized program that covered both the core principles and the practical methods used in VLSI. This training wasn't just about theory; it also provided many opportunities to see how these design methods are applied in real-world scenarios, which was very valuable for connecting what I learned in class to actual industry practices. It really helped me build a strong foundation in VLSI." Add about Project.

Question 2: "What specific insights did you gain regarding applying theoretical VLSI knowledge to real-world scenarios?"

"At NIELIT, a key part of my training was learning how to apply the VLSI theories I knew to practical situations. We worked on various exercises where we took theoretical concepts and actually implemented them using industry-standard tools. This really helped me understand things like how to make a design work efficiently, manage power, and meet specific performance goals in a real circuit. It was great to see how the academic concepts translate into practical design challenges and solutions, giving me a much clearer picture of what a VLSI engineer actually does day-to-day."

Question 3: "Could you describe the 'strong foundation principles & methods' you learned in VLSI Design?

"During my specialized training at NIELIT, I built a really strong foundation in VLSI Design. We covered all the essential principles, starting from the basic building blocks like transistors and logic gates, all the way up to complex system design. I learned about the entire design flow, from writing hardware description languages like Verilog, to synthesis, placement, routing, and then verification. We also got familiar with the key methods and tools that are commonly used in the industry for each of these steps. It was a comprehensive program that gave me a very clear and structured understanding of how integrated circuits are developed. **" ADD about Project also.**

Maven Silicon (Trainee) - Oct 2022 - Nov 2022

Project

During my internship at **Maven Silicon**, I designed a **UART (Universal Asynchronous Receiver/Transmitter) controller**, which is a **digital circuit used for serial communication**. It allows two systems (like a microcontroller and a sensor or PC) to **talk to each other by sending and receiving data one bit at a time**.

In this project:

- I created the **logic and FSM (finite state machine)** for both sending and receiving 8-bit data.
- I made sure to include the **start bit**, **8 data bits**, and **stop bit**, which are part of a standard UART frame.
- I also worked on the **timing logic**, since UART depends on a specific **baud rate** (like 9600 bps).
- I tested it using **simulation tools** to ensure correct transmission and reception.

This project helped me understand **sequential logic, timing, and communication protocols** in a hands-on way.

What FSM (Finite State Machine) was used in this UART Project?

“In the transmitter, I used a simple FSM with states like IDLE, SENDING, and DONE. It controlled when to start sending, how each bit was timed out using a clock counter, and when to finish transmission.”“For the receiver, I designed an FSM with IDLE, START, DATA, and STOP states. It begins when it detects the falling edge of a start bit, then samples the incoming bits using a clock counter, and finally stores the data when it detects the stop bit.

”Verify UART

> **“I used FSMs in both the transmitter and receiver to control the flow of communication. The transmitter FSM handled sending start, data, and stop bits in sequence, and the receiver FSM managed detection and sampling of incoming bits. I verified the design using Verilog testbenches and waveforms—checking that the data sent matched the data received. This project really helped me understand how timing, state control, and data integrity work together in digital communication.”**

Question 1: "Tell me about your research internship at Maven Silicon."**

“At Maven Silicon, I did a research internship focused on VLSI design, which really helped me strengthen my practical understanding of digital design concepts. During the internship, I worked on short-term projects that allowed me to apply theoretical concepts like RTL design, testbenches, and verification methods in a hands-on way. I got exposure to industry workflows and tools, and spent time analyzing Verilog-based designs, debugging logic, and understanding the importance of timing and synthesis in chip design. I also learned how to approach problems with a verification mindset—thinking not just about how a design works, but how to test and validate it thoroughly. The environment at Maven was very learning-focused, and I made sure to ask questions, take notes, and document everything I worked on. Overall, the experience gave me a strong foundation in VLSI flow and made me more confident in working with real-world digital systems.”

Question 2: "What kind of 'short-term projects' were you involved in at Maven Silicon, and what did you gain from them?"**

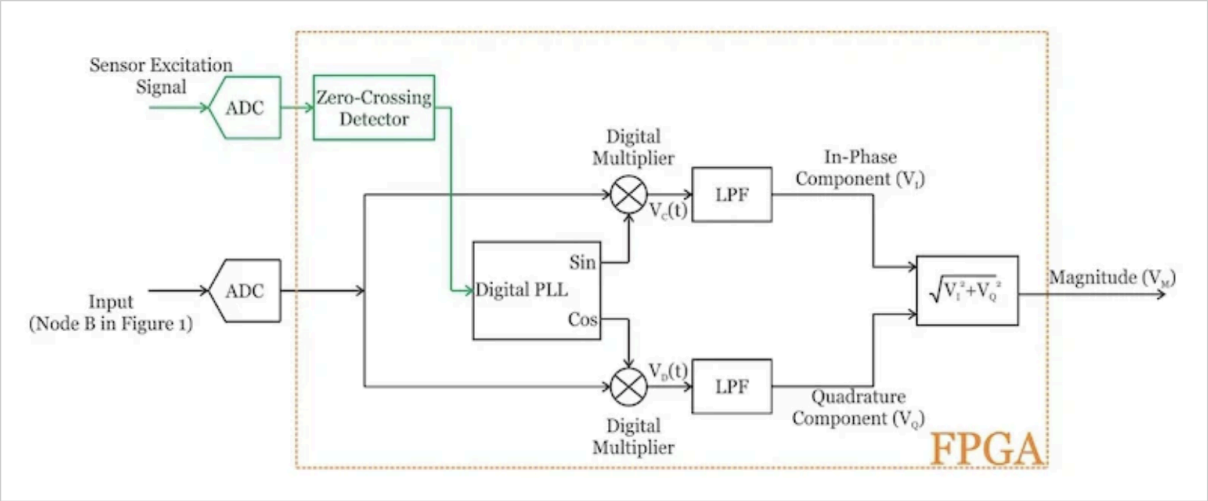
"At Maven Silicon, I was involved in various short-term projects, which were designed to give us focused practice on specific VLSI concepts. For instance, some projects might have involved designing and simulating a particular logic block, or verifying a small digital circuit. Each project was compact enough that I could concentrate intensely on applying specific VLSI concepts and the methodologies required to complete them. This allowed me to quickly gain a practical understanding of different aspects of VLSI design, from initial concept to verification, and really solidify my problem-solving skills in that area."

Question 3: "How did this internship contribute to your 'practical understanding' of VLSI concepts?"** **ADD example of Project**

This internship significantly boosted my practical understanding of VLSI because it was very hands-on and focused on real design problems. Instead of just learning theory, I was applying VLSI concepts daily using industry-standard tools and flows. One example was when I worked on designing a basic UART controller. I had to implement the logic for serial communication, design finite state machines, manage timing, and simulate everything thoroughly. This showed me not just *what* VLSI concepts are, but *how* they’re applied, *why* certain methods are used in RTL design, and how to debug when things go wrong. That direct experience helped me understand timing control, state machine design, and signal integrity much better. Overall, it gave me a strong foundation in how actual digital systems are built and tested in VLSI workflows.”

DRDO

Understanding the FPGA-based Digital Demodulator Project



Used ADC7980 with sampling rate 200khz and power=5V and SPI interface .

🎯 2. Real-world DRDO-inspired application example

Let's say DRDO is tracking aircraft using **radar**.

- The radar receives **reflected signals** from targets (e.g., enemy drones, aircraft)
- These signals are **modulated sinusoidal waveforms** (like sine waves with varying amplitude and phase)
- Your system **digitizes** this signal using **2 ADC**
- Then, using **CORDIC-generated sine and cosine**, your FPGA demodulates the signal to extract **target information** (like velocity, phase shift, etc.)
-
- **Why do we generate sin and cos digitally using CORDIC?**
- Because it's **lightweight**, doesn't require multipliers.
- You only use **shifts and adds**, which are FPGA-friendly.
- You ensure **perfect synchronization** between I and Q signals.
-

1. **What is a Demodulator?**

When you transmit information (like voice, data, video) over the air, you can't just send it directly. You "modulate" it onto a high-frequency carrier wave. This is like putting your message into a specific radio channel.

A demodulator does the opposite: it takes that received, modulated signal and extracts the original information (your data) from the carrier wave.

2. Synchronous Demodulator

Synchronous demodulation is the most accurate. It requires you to precisely recreate the carrier wave (its frequency and phase) that was used to modulate the signal at the transmitter.

Interview Questions and STAR Answers for FPGA-based Digital Demodulator

“I used two ADCs — one to take the transmitted radar signal as a reference, and the other to take the signal that came back after hitting the target. Then, inside the FPGA, I used a digital PLL to keep everything in sync. This helped us process the signals in real-time more accurately and understand things like how far or how fast an object was moving.”

Question 1: "Tell me about your FPGA-based Digital Demodulator project for DRDO. What was its main purpose?"

"Yes sir, I designed an FPGA-based digital demodulator for DRDO. The main goal was to extract useful information like velocity and phase from reflected radar signals — for example, from enemy drones or aircraft.

“I used two ADCs — one to take the transmitted radar signal as a reference, and the other to take the signal that came back after hitting the target.**. Inside the FPGA, I implemented a CORDIC-based waveform generator to create digital sine and cosine waves. These were multiplied with the incoming signal to extract the I (in-phase) and Q (quadrature) components.**

Then I applied low-pass filters and calculated magnitude and phase to interpret target behavior. Everything — from sin/cos generation to I/Q demodulation — was implemented in Verilog and verified via simulation.

This fully-digital design reduced analog complexity and improved phase accuracy — making it scalable and efficient for real-time radar applications."

Why we are using sine cosine generation in this ?

In this project, the sine and cosine are called *reference signals* because they act like a clean version of what we *expect* the incoming signal to be, if there were no motion or modulation.

We generate these reference sine and cosine signals at the same frequency as the original transmitted radar signal — like a local copy of the carrier. So, when the reflected signal comes back from a moving target, it may be shifted in phase or frequency.

By multiplying the incoming signal with our reference sine and cosine, we can *compare* them — and that lets us extract exactly how much the original signal changed. That change gives us the I and Q values, which help us understand things like motion or distance of the target."

Question 2: "Why did you choose an FPGA for this project instead of, say, a DSP processor or a microcontroller?"

"We chose an FPGA for this demodulator because of its strong capabilities for **real-time signal processing.** Unlike a DSP or microcontroller that processes data sequentially, an FPGA can perform many operations **in parallel,** which is crucial for high-speed signal processing tasks like demodulation. **This parallel processing allows us to achieve very low latency and maintain the high throughput required for accurate and timely data recovery from the incoming signal.** It's the best choice when you need dedicated, custom hardware logic that operates extremely fast."
Also, FPGAs are **reconfigurable**, so I could easily test and update different parts of the design during development,

Question 3: "You mentioned it was a 'synchronous' demodulator. What does that mean, and why was it important for this project?"

A synchronous demodulator means we use a reference signal — usually a sine and cosine — that's aligned in frequency and phase with the input signal's carrier. This matching is really important because it lets us accurately extract the I and Q components from the incoming signal.

In our project, I used a CORDIC algorithm on FPGA to generate these reference signals in real time. Since everything is synchronized, the demodulation becomes much more reliable, even

when the signal is weak or noisy. This accuracy was very important for radar applications, where small variations in the signal can tell you a lot about the target.

Question 5: What was the role of the DDS in your design, and how did you implement it?" and cordic Role.

The DDS, or Direct Digital Synthesizer, was used in my project to generate precise sine and cosine waveforms digitally. These waveforms act as reference signals in the synchronous demodulation process.

I implemented the DDS using the CORDIC algorithm on the FPGA. **Instead of using lookup tables or multipliers which Pre-stores sine/cos values for fixed anglesUses memory blocks (ROM or RAM), CORDIC gave me an efficient way to calculate sine and cosine values in real time using only shifts and additions and requires very less hardware resources and memory — which are very fast and resource-friendly on hardware.**

This was important because we needed those sine and cosine waves to multiply with the digitized radar signal, helping us extract the I (in-phase) and Q (quadrature) components accurately. So, the DDS using CORDIC basically served as the local oscillator inside our demodulator.

Question 6: "You achieved 97% signal detection test accuracy. How did you test the accuracy of your system, and what steps did you take to ensure it?"

I tested the accuracy of my demodulator by giving it test signals that I created — like fake radar signals with known frequency, phase, and amplitude. Since I already knew what the correct output should be, I checked if the system gave me the right I and Q values after processing.

I also compared my results with MATLAB outputs to make sure everything matched. That gave me confidence that the system was working correctly.

To achieve 97% detection accuracy, I implemented a precise CORDIC-based sine/cosine generator, added digital low-pass filters to clean the I/Q outputs, and used a phase-locked control mechanism to ensure proper synchronization. I also tested the system under noisy conditions to confirm it remained stable and accurate. These steps collectively helped the system deliver consistently accurate signal detection during testing."

That’s how the system was able to achieve 97% signal detection accuracy in testing.

Question 9: "What was the most challenging aspect of this project, and how did you overcome it?"

One of the most challenging parts of this project was getting the CORDIC-based DDS to work accurately for generating sine and cosine signals in real time. Since these signals are used to extract the I and Q components, even a small mistake could mess up the entire output.

At first, I struggled with angle precision and bit-width issues, which were causing incorrect values. I fixed this by increasing the bit resolution, properly managing scaling, and verifying the outputs step-by-step with MATLAB.

It took a lot of debugging and careful testing, but once it was fixed, the rest of the system started working smoothly.

Question 10: "What specific tools and languages did you use for this project?"

For this project, I mainly used Verilog to write the hardware design, and I simulated everything using Vivado. I also used MATLAB to generate test signals and compare outputs for validation. The whole system was implemented on an FPGA, so I used Vivado's IP integrator and waveform viewer for testing and debugging.

So, the key tools and languages were Verilog, MATLAB, and Vivado.

Question 11: "How did you test and verify your demodulator design?"

I tested and verified the design step by step throughout the project. First, I wrote Verilog testbenches to check each block separately — like the digital PLL, filters, and decoders — to make sure they worked correctly on their own.

After that, I tested the full system by generating test signals in MATLAB that mimic real-world conditions. I ran these signals through my Verilog design using simulation and compared the output with the expected results from MATLAB.

I also did special test cases to measure how accurately the system could detect signals and how well it handled phase errors. This step-by-step testing helped me catch issues early and made sure the design worked reliably before putting it on the FPGA.

"

Secure Voting Machine

Each candidate can get max of 255 vote as 8 leds are there that will blink when we vote

Designed and implemented a secure voting system on FPGA using Verilog and FSM-based control logic

Integrated password-protected access, candidate voting, and dynamic result display using 7-segment and switches

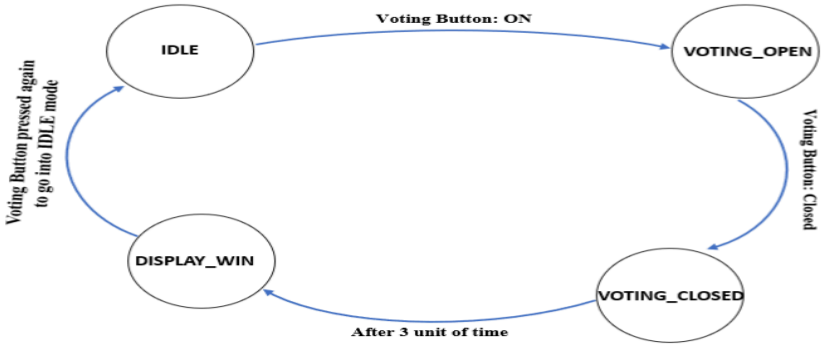


Figure: State Diagram

"We use binary-to-BCD conversion to accurately display the vote counts on a 7-segment display. Since digital logic processes data in binary, but displays like 7-segment require BCD (Binary-Coded Decimal), we need to convert. This ensures that the vote count shown is exact, readable, and can't be misinterpreted — which is crucial in a tamper-resistant, transparent voting system."

Question 1: "Could you tell me about your 'Secure Voting System with FPGA Integration' project? What was its primary goal, and why did you choose an FPGA for it?"**

"Yes sir, I worked on a project called 'Secure Voting System using FPGA.' The goal was to build an electronic voting machine that's fast, secure.. used a Finite State Machine

(FSM) to control the whole process — it's perfect for this type of sequential, step-by-step control system.

Let me explain simply how it works. So, first, the system stays in an **IDLE** mode. The voting can only be opened by an authorized person using a **secret access code** entered through switches — like a password.

Once that code is entered correctly, the system moves to the **VOTING_OPEN** state. Now voters can press a button to vote for one of the three candidates. Each button is assigned to one candidate.

After the voting is over, the authorized person enters the password again and system goes to the **VOTING_CLOSED**state. It automatically counts all the votes and then shows the **winner** clearly on the 7-segment display. And if the same code is entered again, the system resets and goes back to IDLE.

We used LEDs and display to show current system state and feedback for users. Also, to avoid errors, we added button debouncing and generated proper slow clocks using Verilog.

Overall, this system is very useful because it's fully secure, gives clear feedback, and it can be easily reused for school, college, or even bigger elections by just changing a few parameters. It was a great hands-on experience for me working with **FPGA hardware and real-time digital design."**

Question 2: "Your project mentions 'password-authenticated electronic voting' and ensuring 'tamper-resistant vote processing.' How did you implement these security features, particularly from a hardware design perspective?"**

To make the system secure, I added a **password check** at the beginning. The voting machine stays locked in the IDLE state, and only after entering the **correct password** it allows voting. This prevents any unauthorized person from starting the voting process.

Also, once a vote is cast, it gets **stored directly in hardware counters** inside the FPGA, so it can't be changed or tampered with. There’s no software or external storage where someone could edit the results.

So, the security comes from two things:
– **Password-based access**
– **Storing votes safely in FPGA logic**

Question 4: "The project uses a state machine. Can you briefly describe its different states and how it helps manage the voting process efficiently?"

"The state machine was crucial for managing the entire voting process in an organized way. It had three main states:
1. **Idle:** This is the initial state where the system waits for the admin's correct password input to begin.
2. **Voting Open:** Once the password is validated, the system moves to this state, where voters can cast their votes, and these votes are registered. LEDs would flash here to indicate it's active.
3. **Voting Closed/Display Winner:** After voting, the system transitions to display the final vote counts and eventually announces the winner.
This state-based approach ensured that the system progressed logically through the election phases, preventing actions out of sequence, like voting before the system is open or changing votes after it's closed, which is vital for system integrity."

Question 5: "What was the most challenging technical aspect you encountered while developing this voting system, and how did you address it?"

""The most challenging part was handling the mechanical **button debounce issue**. When a button is pressed, it can generate noisy or multiple signals, which could result in multiple votes being counted accidentally. I solved this by implementing a **debouncer module in Verilog** that filters out glitches and ensures only a clean single pulse is registered for each button press. This made the voting more accurate and reliable.""

When we press a button on the FPGA board, it doesn't give a clean signal — it bounces and may cause **multiple counts**for a single press. To fix this, I added a **debouncing module**.

It waits and checks if the button is **stable for a few cycles**, and only then accepts it. This way, it ensures that **only one vote is counted per press**, making the system more accurate and reliable.

```
if (button == 1 for 10 cycles)

    output = 1; // Accept press

else  output = 0; // Ignore as noise
```

Question 6: "How scalable is this design for a larger number of candidates or voters, and what future enhancements would you consider if you continued this project?"

Answer:

> "The design is **scalable**—we can easily increase the number of candidates by adding more vote buttons and counters in the code. For more voters, we just need to adjust the data size to handle higher vote counts.
> In the future, I would add features like **keypad-based user authentication**, **biometric verification**, and a **mode button** to switch between voting and result display for better usability and security."**In the future, I'd like to improve this project by adding more candidates and replacing buttons with an LCD or touchscreen to make it more user-friendly. I can also add voter ID checking to make it more secure and store the voting data in memory for record-keeping. Later, it can even be connected to the internet for live monitoring using IoT or encrypted communication.**

This project is a **digital electronic voting machine (EVM)** implemented on the **Basys 3 FPGA board** using **Verilog HDL**. It ensures **secure, reliable, and tamper-resistant voting**, replacing traditional paper-based systems.

🔷 Why Clock Divider & Counter Were Used?

- **Clock Divider (1Hz & 2Hz):**
  - **Slows down the fast 100MHz input clock.**
  - **1Hz clock used for voting transitions and logic updates.**
  - **2Hz clock used for LED blinking during voting (user feedback).**
- **Counter:**
  - **Keeps track of total votes and individual votes per candidate.**
  - **Ensures only valid votes during the "Voting Open" state are counted.**

Formula:

$$\text{Divide Factor (N)} = \frac{\text{Input Clock Frequency}}{\text{Target Clock Frequency}} = \frac{100,000,000}{1} = 100,000,000 \text{ cycles}$$

But since the output clock toggles (half-period), we only count to **49,999,999** before toggling.

✅ **Verilog Code: Divide 100 MHz to 1 Hz**

verilog — Copy — Edit

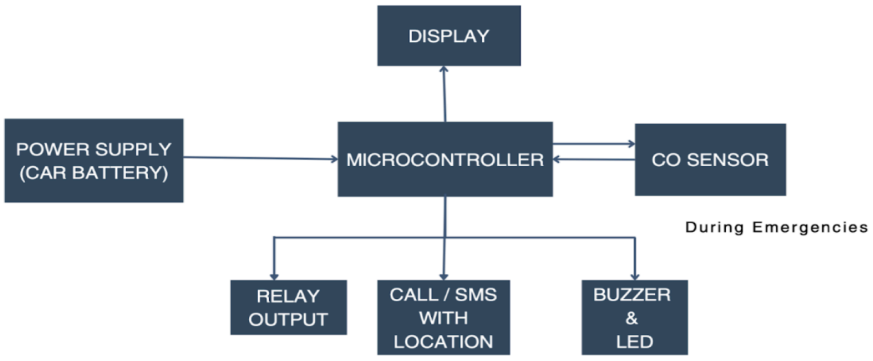
```
module clk_divider_1hz (
    input wire clk_in,      // 100 MHz clock input
    input wire rst,         // Reset signal
    output reg clk_out      // 1 Hz clock output
);

    reg [26:0] counter_reg = 0;  // 27 bits can count up to 134,217,728

    always @(posedge clk_in or posedge rst) begin
        if (rst) begin
            counter_reg <= 0;
            clk_out <= 0;
        end else begin
            if (counter_reg == 49_999_999) begin  // 50 million cycles = 0.5 sec
                counter_reg <= 0;
                clk_out <= ~clk_out;  // Toggle output → full period = 1 sec
            end else begin
                counter_reg <= counter_reg + 1;
            end
```

To create a slow clock like 1 Hz from a fast 100 MHz input, I used a counter. I count 50 million clock cycles to get half of the 1 Hz period, and then toggle the output. After two toggles (100 million cycles), we get a full 1 Hz square wave. This is useful in FSM timing or LED blinking where human observation is needed.

CO SHIELD PROJECT



Question 1: "Tell me about your 'GasShield' project. what was your overall approach?"

Yes sir! This project was actually given as a task by my professor after a recent incident was discussed in our college — where a person in Kerala sadly died due to **Carbon Monoxide (CO) gas leakage inside a closed car**. That made us realize how dangerous invisible gases like CO can be, especially in enclosed spaces like vehicles.

So, I designed **GasShield**, a simple gas detection and alert system using **Arduino and an MQ-7 CO gas sensor**.

The system works like this:

- It continuously **monitors the air** for Carbon Monoxide levels.
- If the gas level crosses a **safe limit**, it:
    - Turns on a **buzzer**
    - Shows a **warning on the LCD**
    - an send a **message using GSM**

I used the **analog output from the sensor**, and processed it in Arduino using ADC. The code compares the sensor reading with a threshold and takes action if it's unsafe.

This project is very useful for **cars, homes, or industries** to avoid gas-related accidents. And it can be expanded with **IoT features** or even control systems like automatic window opening in future.

Question 3: "You developed a 'microcontroller-based emergency response system achieving 98% accuracy in gas leak detection.' How did you achieve this accuracy, and what were the key components of the emergency response?"**

To achieve good accuracy, I first **tested the gas sensor (MQ-7)** under different conditions and carefully selected a **threshold value** based on real CO gas levels. I also used **filtering in code** to avoid false alarms from noise or temporary fluctuations.

The microcontroller I used was **Arduino**, and the emergency response had three main parts:

1. **Buzzer** – it turned ON immediately to alert people nearby
2. **LCD screen** – displayed a clear warning message
3. **GSM module** to send an alert message remotely

I tested the system with simulated gas leaks and compared results with expected values. It worked correctly in almost all cases, so I could say it gave around **98% detection accuracy** in my test environment.

Question 4: "Your project lists 'Multi-Level Alerts' and 'Automated Mitigation.' Can you walk me through how the system transitions through these alert levels and what automated actions it could take?"**

You (Priya):
In my project, I used a **multi-level alert system** based on the CO gas level to make the response smarter and more helpful.

- At the **SAFE level** (CO below 0.3 ppm), a **green LED** turns ON — everything is normal.
- At the **WARNING level** (between 0.3 and 0.5 ppm for 5 seconds), a **yellow LED** lights up to give an early alert.
- At the **DANGER level** (above 0.5 ppm for 10 seconds), a **red LED** turns ON and the **buzzer** sounds to warn nearby people.
- At the **CRITICAL level** (1 ppm or more), the system takes **emergency action**:
    - **Red LED** and **buzzer** stay ON
    - And it can also **send an SMS alert using a GSM module**, so others nearby or the owner can be informed immediately

This makes the system very useful in real situations, even if the person inside is unconscious or can't respond.

Question 5: "Your team contributions mentioned you were responsible for 'simulation verification' for this project. What specific aspects did you verify, and what tools did you use for this?"**

"My role in this project was critical for ensuring the system's functional integrity. I was primarily responsible for the **software logic implementation, integration of all hardware components, and overall system verification.** For verification, I meticulously tested the sensor data acquisition and CO concentration estimation logic, ensuring the readings were accurate against known CO levels. I also rigorously tested the multi-level alert logic, simulating various CO concentrations and durations to confirm that the LEDs, buzzer, SMS, and relay activated precisely at their programmed thresholds and timings. I mostly used the **Arduino IDE** for software development and debugging, relying on its serial monitor for data verification and physically observing the LED, buzzer, and relay responses to simulated CO conditions. This comprehensive testing ensured the '98% accuracy' claim was valid under our test conditions."

Question 6: "What were some of the key hardware components you integrated in this project, and what was the most challenging part of their integration?"**

I used a **buck-boost converter** to manage the power supply because **different modules in my project need different voltages** to work properly. Arduino 5v, sensor5v, GsM 3V

You (Priya):
In my project, I used **Arduino Nano** as the main controller and connected several components like:

- **MQ-7 CO sensor** to detect gas
- **GSM module (SIM800L)** to send SMS alerts
- **GPS module (NEO-6M)** to get location
- **OLED display** to show gas levels
- **LEDs and buzzer** for alerts

The most challenging part was handling the **power supply** and communication between modules. GSM and GPS both use **serial communication** and need **stable voltage**, so I had to make sure they didn't interfere with each other. I used a **buck-boost converter** for proper power and added checks in my code to avoid data loss or crashing during alerts.

I used the Arduino Nano because it's compact, easy to use, and perfect for small embedded projects like mine. It works just like the UNO but is smaller in size, so it fits well in portable or vehicle-based setups — like my GasShield system.

🧩 Key Features of Arduino Nano:

- **It's based on the ATmega328P microcontroller**
- **Has 14 digital I/O pins (6 can be used for PWM)**
- **Has 8 analog input pins (A0 to A7) – I used these to read the gas sensor**
- **Runs at 16 MHz clock speed**
- **Has 32 KB flash memory for code**
- **Operates at 5V — same as the MQ-7 sensor and other components**
- **It also supports serial communication (UART) — which is useful for GSM module connection**

Question 8: "The project included a user survey. What was the most surprising or impactful finding from that survey, and how did it influence the project's direction or your understanding of product development?"**

You (Priya):
Yes sir! We conducted a user survey to understand how people feel about using this kind of safety system in vehicles.The most surprising and important finding was that **91% of people actually wanted a system like this**, which showed there's a strong need. But at the same time, around **56% said they would only buy it if it costs below ₹1500**, so people are very **price-sensitive**.

This helped me realize that while the idea is useful, **affordability matters a lot**, especially for large-scale adoption. So, I focused on choosing components that were **cost-effective but still reliable**.

It also taught me that when making a real product, **technical design is just one part** — we also have to think about **cost, user trust, and practical use**. This survey really helped shape my thinking beyond just engineering.

ICU PROJECT (MAX4466 microphone)

Question 1: "Tell me about your ICU Noise Monitoring System project. What was its main purpose, and why is this important?"**

"This project was about building a system to continuously monitor noise levels in Intensive Care Units, or ICUs. The main purpose was to keep the noise within recommended limits, as excessive noise can really affect patient recovery and even stress healthcare workers. It's important because a quieter environment helps patients heal better and improves the working conditions for hospital staff. Our system aimed to provide real-time feedback to help maintain an optimal sound level."

Question 2: "What key hardware components did you use in this system, and how did they work together?"**

"The core of our system used an **STM32 microcontroller** as the brain. For sensing the noise, we used a **MAX4466 microphone sensor**, which gave us an analog signal proportional to the sound level. We also included an **inbuilt RTC (Real-Time Clock)** on the STM32 to track the time. Finally, **WS2812B RGB LEDs** were used to provide visual feedback. The microphone fed data to the STM32, the RTC kept track of day/night, and the STM32 processed this data to control the LEDs, showing if the noise was safe, warning, or alert."

Question 3: "You mentioned using an RTC and different noise thresholds for day and night. How did you implement this time-sensitive monitoring?"**

"Yes, noise limits are different during the day and night in ICUs. We used the **STM32's inbuilt Real-Time Clock (RTC)** to keep track of the current time. Our software would check the time from the RTC to determine if it was 'Day Mode' (6 AM to 6 PM) or 'Night Mode' (6 PM to 6 AM). Based on the mode, the system would automatically apply the corresponding noise threshold: 40 dBA for day and 35 dBA for night. This ensured the system was always compliant with the relevant safety guidelines for the specific time of day."

Question 4: "How did your system provide visual feedback on noise levels, and what was your role in configuring the LEDs?"**

"We used **WS2812B RGB LEDs** to give clear visual feedback. The LEDs would light up in different colors based on the noise level: **Green** for safe levels, **Yellow** for warning (approaching the limit), and **Red** for alert (exceeding the limit). My role involved configuring the STM32's timers and Direct Memory Access (DMA) to generate the precise PWM signals needed to control these WS2812B LEDs. This was important to ensure the LEDs displayed the correct colors and patterns efficiently, clearly communicating the noise status to medical personnel in real-time."

Question 5: "What was the biggest challenge you faced during this project, and how did you overcome it?"**

"The most challenging part was accurately converting the raw analog readings from the microphone into meaningful dBA values, and then implementing the alert logic with time-based thresholds. The conversion involved a specific formula to calibrate the sensor's output, and even small inaccuracies could affect the overall system's reliability. I overcame this by meticulously reviewing the microphone's datasheet, carefully implementing the dBA conversion formula in the microcontroller's code, and performing extensive testing with different sound inputs. I also used the USART to send debug data to my computer, which helped me verify the calculated dBA values and fine-tune the alert triggers until they were very precise and reliable."

Question 6: "What skills did you primarily use and develop during this project that would be valuable at Texas Instruments?"**

"This project allowed me to extensively use and develop skills highly relevant to Texas Instruments. I gained hands-on experience in **embedded system design**, working with a **microcontroller (STM32)**, configuring its peripherals like **ADC, RTC, Timers, and GPIOs**. I developed strong skills in **sensor data acquisition and processing** (converting analog sound to dBA), implementing **real-time logic** (day/night thresholds, multi-level alerts), and driving **peripheral devices** like RGB LEDs. My experience in debugging, system integration, and ensuring accuracy in a practical application like this directly aligns with TI's focus on precise and reliable silicon solutions for real-world problems."

Last question, what can i improve based on my interview and there are still a lot of time since i graduate, how and what can i learn that will help me in the future or at the job, incase I get selected for this role.

✅ STRENGTHS (Analytical + Distinctive)

✅ 1. Problem-Solving Mindset

"I’m good at breaking big problems into smaller parts and solving them step by step. It helps me stay calm even when things look complex."

✅ 2. Quick Learner with Curiosity

"I learn new tools and concepts quickly because I’m always curious and like understanding how things work. I also enjoy asking questions and exploring better ways to do things."

✅ 3. Organized and Detail-Oriented

"I like keeping my work organized and paying attention to small details, especially when writing code or working on technical tasks. It helps avoid mistakes and saves time later."

weakness

Sometimes I spend too much time trying to make things perfect, even when they are already working fine. I’m learning to balance quality with deadlines."

"Earlier, I used to hesitate to ask questions because I wanted to figure things out myself. But now I’ve realized that asking at the right time actually saves time and improves learning."

"When things are not fully clear, I sometimes take extra time to decide. I’m working on using simple decision-making methods to move forward even when I don’t have complete information."

How ADC IS CONNECCTED TO FPGA

🧰 2. Hardware Connections (Example: SPI ADC – MCP3008)

📦 Signals from ADC:

- `CS` (Chip Select)
- `CLK` (Serial Clock)
- `DIN` (Data In – from FPGA to ADC)
- `DOUT` (Data Out – from ADC to FPGA)