11310CS210401

Hardware Design and Lab

Prof. Chun-Yi Lee

Final Project

Team-17 Report:

Design and Implementation of a Simple Radar System Using Ultrasonic Sensors, Servo Motors, and VGA Display

Submitted by:

Aryabima Mandala Putra/黃峰/111006240

Vincent Jovian Christanto/黃財城/111006222

7 November 2024

Table of Contents

Introduction

Radar systems are widely used in various applications, including navigation, weather monitoring, and object detection. In this project, we designed and implemented a simple radar system using an FPGA. The system integrates an ultrasonic sensor, a servo motor, and a VGA monitor to visualize detected objects in real-time. By combining FPGA technology with hardware components, this project demonstrates a compact and efficient radar solution suitable for educational and experimental purposes.

The system works by rotating the ultrasonic sensor with the servo motor to scan the surrounding environment. The sensor measures the distance of objects, and the FPGA processes the data to display it on a VGA monitor. This real-time visualization provides a clear representation of the environment, making it easier to analyze object positions and distances.

Motivation

This project idea came from our last project, where we worked on a car that utilizes an ultrasonic sensor for when-to-stop detection. During that time, we thought about using the ultrasonic sensor again for our final project and came up with an item-detecting system. Later, we decided to improve the idea and turn it into a radar.

At first, we didn’t plan to use a VGA monitor because we weren’t familiar with it. But we realized the challenge would make the project more interesting, so we started learning how to work with VGA. In the end, figuring out the VGA part became the toughest but most exciting

part of the whole project. It pushed us to learn something new and made the project more complete.

System Specification

The radar system is designed using an FPGA board, combining an ultrasonic sensor, a servo motor, and a VGA monitor for real-time object detection and visualization. The system integrates both hardware and mathematical operations to achieve accurate and efficient performance. Below are the detailed specifications:

- **FPGA Board**: Acts as the central processor, controlling the servo motor, processing distance data from the ultrasonic sensor, performing mathematical calculations, and generating VGA signals for display.
- **Ultrasonic Sensor (HC-SR04)**: Measures the distance of objects by emitting ultrasonic waves and receiving the reflected signals. The measured data is sent to the FPGA for further processing.
- **Servo Motor**: Rotates the ultrasonic sensor in a sweeping motion, covering a wide range of angles for environmental scanning. Each step corresponds to a specific angle controlled by the FPGA.
- **VGA Monitor (640x480)**: Displays the radar output in real time, visualizing detected objects on a graphical interface. The display uses Cartesian coordinates to show the relative position of objects.
- **Mathematical Operations**:
    - The system converts polar coordinates to Cartesian coordinates *(x,y)* for visualization on the VGA monitor.

    - Equations used: $x = r\ cos(\theta)$, $y = r\ sin(\theta)$ Where r is the distance and θ is the angle of the servo motor.
- **Lookup Table (LUT)**:
    - A precomputed LUT is implemented for sine and cosine values for all possible servo motor angles.
    - This eliminates the need for real-time trigonometric calculations, significantly speeding up the system's performance.
    - The LUT is stored in the FPGA memory, ensuring quick access during runtime.
- **Clock Management**: A custom clock divider ensures proper synchronization of the servo motor, ultrasonic sensor, and VGA signal generation.
- **Power Supply**: All components are powered directly from the FPGA's 3.3V supply, ensuring a compact and efficient design without the need for external power sources.

Key Features:

- **Real-Time Performance**: The radar can detect and display objects in real-time, ensuring smooth and accurate visualization.
- **Optimized Computations**: The use of LUT and pre-calculated trigonometric values enhances computational efficiency.
- **Compact Design**: The system operates entirely on FPGA-provided power, making it portable and self-contained.
- **High Accuracy**: The integration of precise angular control, reliable distance measurements, and efficient calculations ensures accurate object detection.

Experimental Results

Testing Setup

- **Environment**:

  The system was tested in an open space on a flat table to ensure consistent performance. We tested one object at a time, as the system uses a single ultrasonic sensor for detection. A smooth surface was preferred since the ultrasonic sensor is highly sensitive to irregularities in the environment.

- **Tools and Methods**:

  Simple tools, such as boxes and rulers, were used to evaluate the system's detecting capabilities. The measured distances were displayed directly on the FPGA board in centimeters for real-time validation.

- **Testing Materials**:

  Solid objects were used as targets during testing, as the ultrasonic sensor relies on sound wave reflection for detection. To ensure accuracy, objects with materials that absorb sound, such as clothes, cotton, or sponges, were avoided, as these can interfere with the sensor's operation.

Performance Metrics

- **Distance Detection Range**:
  - **Minimum Distance**: Approximately 2 cm.

    - **Maximum Distance**: Adjusted to 240 cm (actual maximum capability of 400 cm, but scaled to 240 cm to match our VGA radar display).
- **Angular Resolution**:
    - The radar achieves a full 180-degree scanning range, as enabled by the 180-degree servo motor.
    - The angular movement is smooth and precise, covering angles from 0° to 180° and back in a continuous cycle.
- **Visual Representation**:
    - The system uses a white beam on the VGA display to simulate the radar's scanning motion. The beam sweeps across the 180-degree range in real time.
    - Detected objects are plotted along the path of the white beam. Since the same scaling and movement calculations are applied to both the white beam and the object visualization, the detected object positions are always accurately aligned with the beam (only moves on the whitebeam-axis).

Observations

Distance Measurement

- **Testing Setup:**

To test the radar's distance-measuring capability, we manually measured the distance of objects using a ruler and compared it to the values displayed on the FPGA's seven-segment display (in centimeters). Below is *Figure 1.1* and *Figure 1.2*

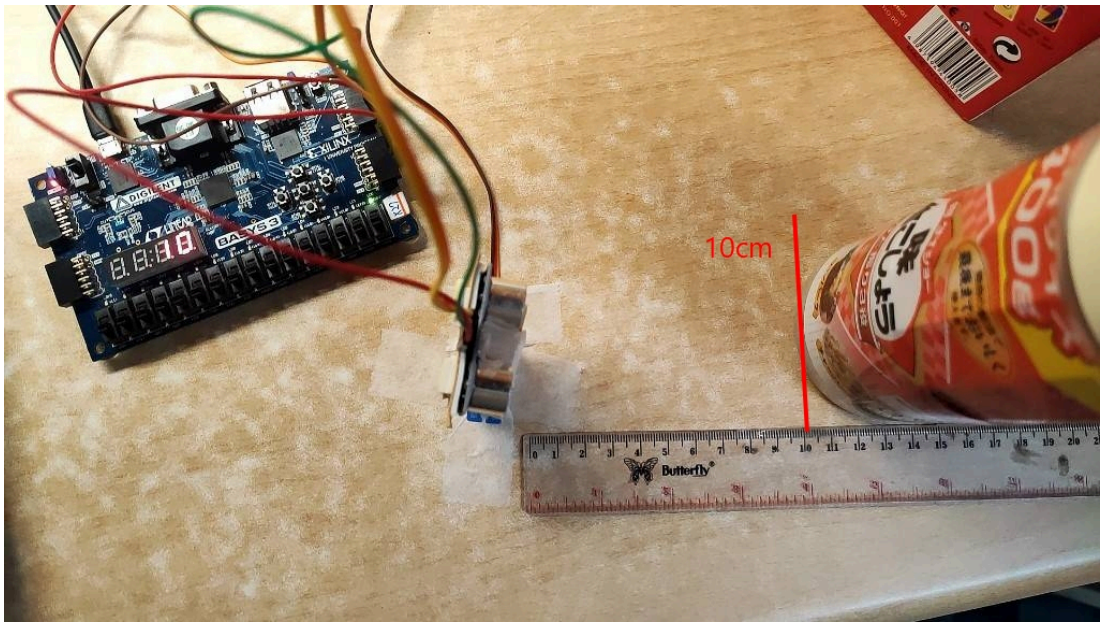

Figure 1.1. Object with range around 10cm from sensor ***(FPGA shows : 10)***

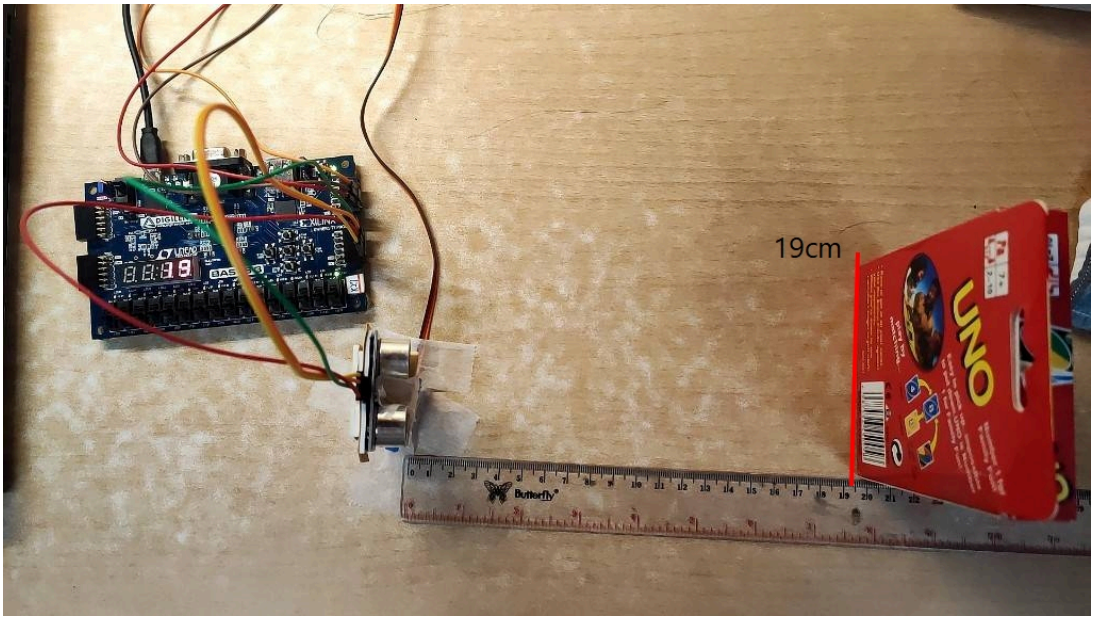

Figure 1.2. Object with range around 19cm from sensor ***(FPGA shows : 19)***

- **Accuracy:**

  The distance measurements showed slight errors, typically the rounding of the values displayed in the fpga may differ from the actual values. This discrepancy is due to the division factor used in the calculation *(5820)* being an approximation for hardware implementation. We use a tool to visualize how we get the scaling calculations, below is *Figure 1.3.*

The formula for calculating distance in centimeters with an ultrasonic sensor is based on the speed of sound and the time measurement from the sensor. Here's the breakdown:

1. Speed of Sound in Air

- The speed of sound is approximately **343 meters/second** or **0.0343 cm/µs**.
- The ultrasonic sensor measures the **round-trip time**, so we divide by 2 to get the actual one-way distance.

2. Distance Formula

The formula for distance in centimeters is:

$$\text{distance\_cm} = \frac{\text{time\_in\_µs} \times 0.0343}{2}$$

3. Clock Dependency

- In FPGA, the `echo_count` represents the time during the ultrasonic pulse in **clock cycles**.
- For a **100 MHz clock**, the clock period is:

$$T_{\text{clk}} = 10\,\text{ns} = 0.01\,\mu\text{s}$$

- The total time in microseconds is:

$$\text{time\_in\_µs} = \text{echo\_count} \times 0.01$$

4. Combining Everything

Substituting $\text{time\_in\_µs} = \text{echo\_count} \times 0.01$ into the distance formula:

$$\text{distance\_cm} = \frac{(\text{echo\_count} \times 0.01) \times 0.0343}{2}$$

Simplify:

$$\text{distance\_cm} = \frac{\text{echo\_count} \times 0.000343}{2}$$

$$\text{distance\_cm} = \frac{\text{echo\_count}}{5820}$$

5. Why 5800?

- The theoretical factor is 5820, but in practice, 5800 is often used as an approximation for simplicity.
- Using 5800 still provides accurate results within an acceptable range.

Figure 1.3. Centimeter Approximation Calculation

VGA Visualization

- **Beam Movement:**

  The radar beam visual is functioning as intended, sweeping forward and backward within a 180-degree range. The beam is designed to simulate the radar's scanning motion in real-time.

- **Synchronization:**
  - The beam's movement is almost synchronized with the actual servo motor rotation, but some hardware challenges remain. Specifically, the servo motor moves slightly slower when transitioning from 180° to around 90°, causing the beam to flicker on the VGA display.
  - To mitigate this issue, we adjusted the synchronization by aligning the beam's speed to match the servo motor's speed, rather than relying solely on directional triggers.
  - The servo motor's PWM signal direction (1 for 0° to 180° and 0 for 180° to 0°) was initially used as the trigger for synchronization. However, better results were achieved by adjusting the speed slightly, ensuring smoother visual performance.
- **Current Status:**

  The synchronization is now stable, and the beam displays smoothly with minimal flickering. This alternative approach has significantly improved the system's visual accuracy and reliability.

Visual Output

The visual output of the radar system is displayed on a VGA monitor, providing a graphical representation of detected objects in real time. The key elements of the display include:

- **Radar Beam**: A white beam simulates the radar's scanning motion, sweeping from 0° to 180° and back.
- **Detected Objects**: Objects detected by the ultrasonic sensor are plotted along the path of the radar beam, based on their calculated distance and angle.
- **Scaling and Movement Alignment**: The system ensures that the detected objects are always accurately aligned with the radar beam, using the same scaling and movement calculations.

Below are some pictures of the VGA monitor.

Figure 1.4. Initial Radar Visualization

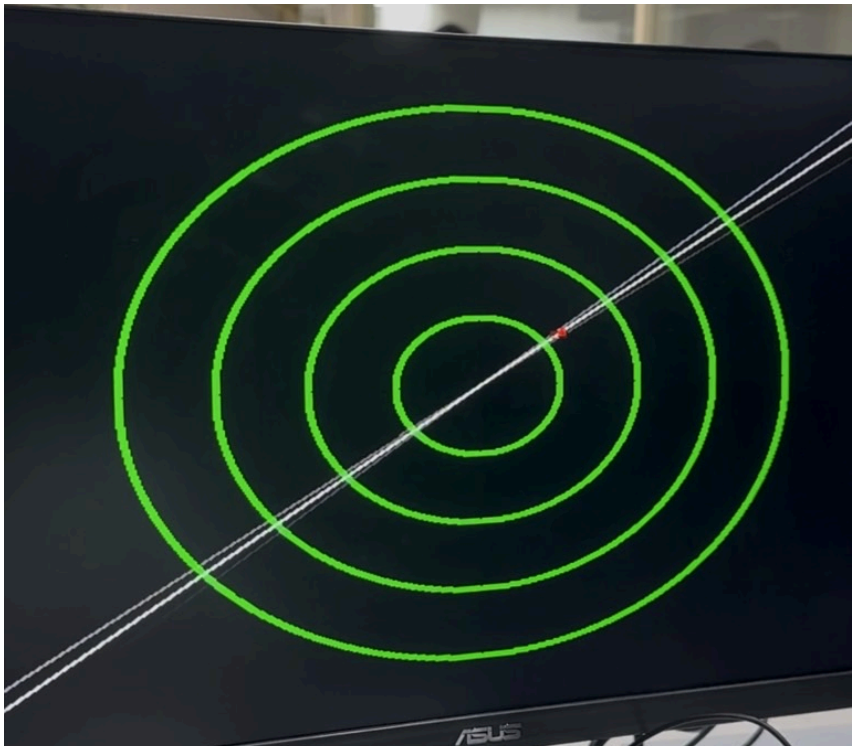

Figure 1.5. Radar Visualization when detecting object (red dot)

In our code, we use four circles, with each circle representing approximately 60 cm in diameter. Since the radar only visualizes the top half (180 degrees), each half-circle corresponds to around 30 cm, giving a total detectable range of 120 cm. Each circle represents an increasing distance:

- **Inner Circle**: Diameter = 50 pixels, approximately 60 cm.
- **Middle Circle**: Diameter = 100 pixels, approximately 120 cm.
- **Outer Circle**: Diameter = 150 pixels, approximately 180 cm.

These circles are scaled to fit the radar's maximum range of 240 cm.

Presentation Question: How does the red dot indicating the object follow the sweeping radar line, while showing the distance to the target?

The radar beam and object are synchronized because both positions (beam_x, beam_y and obj_x, obj_y) are calculated using the same trigonometric functions (cosine_val and sine_val) and the current beam angle. As the beam angle updates, the calculated object position moves in sync with the sweeping line, maintaining alignment.

The object's distance to the radar determines its position relative to the radar center:

A shorter distance_cm places the red dot closer to the center.

A longer distance_cm places it farther out, but within the radar's visible range.

If the object is outside the radar's effective range, it will not appear.

Conclusion

Our radar system that is implemented on an FPGA successfully demonstrates real-time object detection and visualization using an ultrasonic sensor, servo motor, and VGA monitor. The project combines hardware and mathematical principles to achieve a functional and visually interactive system. Below is *Figure 1.6.*

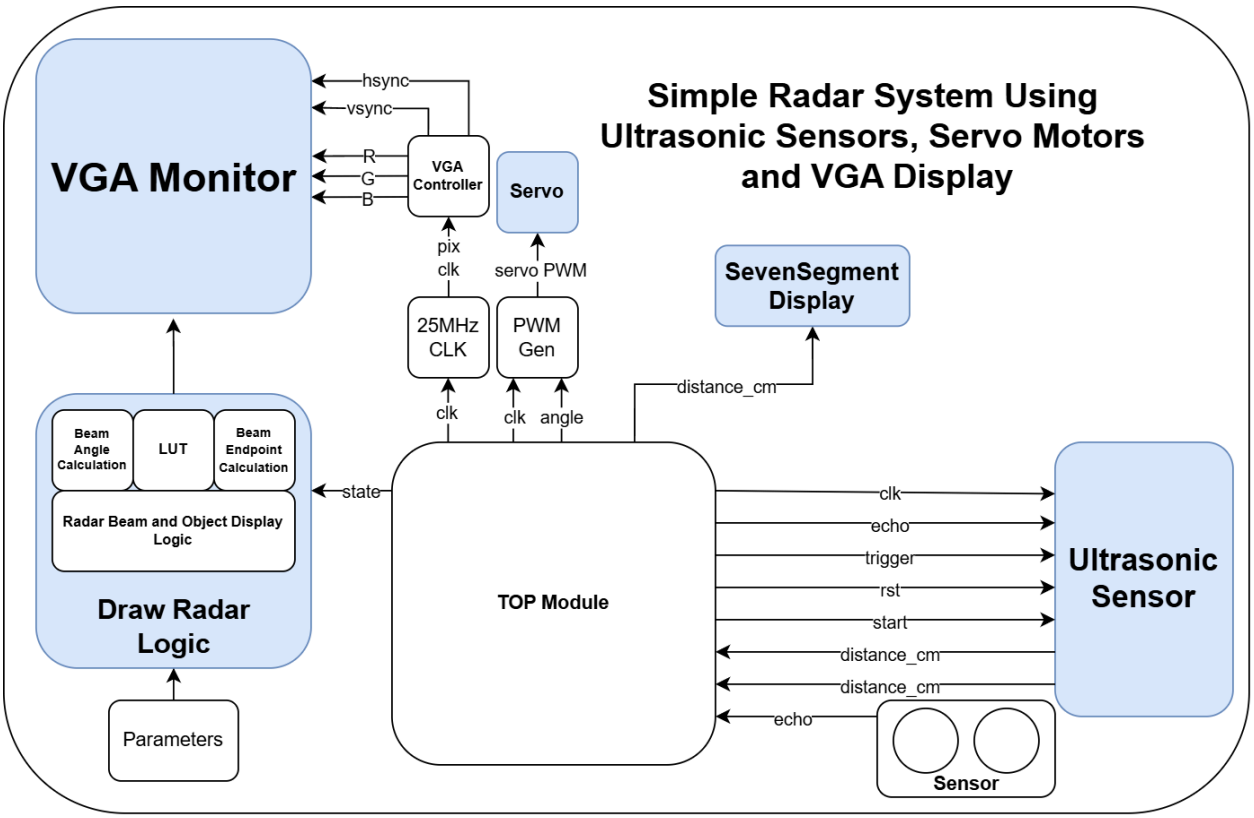

Figure 1.6. Overall Logic Diagram of our Simple Radar

Below is *Figure 1.8.* Is the state diagram of our Ultrasonic Sensor.

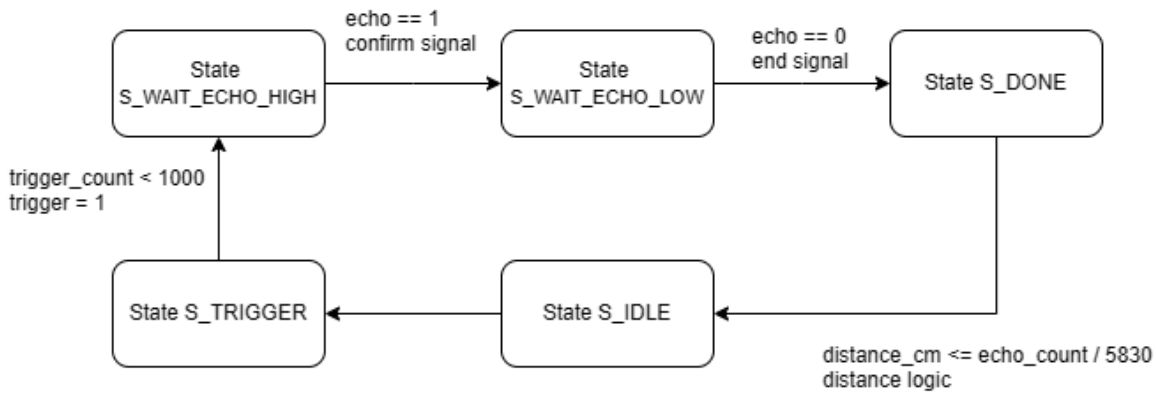

Figure 1.8. State Diagram of Ultrasonic Sensor

Basically, the system obtains the distance from the ultrasonic sensor. This distance is then used to calculate the scaling required for the radar visualization. The visualization logic is handled by the **DrawLogic**, which determines how the data will be displayed. Finally, the VGA controller manages the signals and outputs the visualization to the VGA monitor. The mathematics implementation and other logic that we use for the draw logic are :

Lookup Table (LUT)

- The LUT stores precomputed sine and cosine values for all angles from 0° to 180°.
- These values are used to convert polar coordinates (distance and angle) into Cartesian coordinates *(x,y)* for VGA display.
- Using the LUT eliminates the need for real-time trigonometric calculations, significantly improving computational efficiency. Below how the LUT looks like on our code

```
// ------------------------
// Sine and Cosine Lookup Table
// ------------------------
reg signed [15:0] sine_table [0:359];
reg signed [15:0] cosine_table [0:359];
integer i;
real sine_val_real, cosine_val_real;
initial begin
    for (i = 0; i < 360; i = i + 1) begin
        sine_val_real = $sin(i * 3.141592653589793 / 180.0);
        cosine_val_real = $cos(i * 3.141592653589793 / 180.0);
        sine_table[i] = $rtoi(sine_val_real * 256.0); // Higher precision
        cosine_table[i] = $rtoi(cosine_val_real * 256.0); // Higher precision
    end
end

wire signed [15:0] sine_val = sine_table[beam_angle];
wire signed [15:0] cosine_val = cosine_table[beam_angle];
```

Figure 1.7. Look Up Table for the Cosine/Sine Values.

There is something unique here, the value **256** is chosen to scale the sine and cosine values for efficient computation in hardware because Vivado and most FPGA tools do not natively handle decimal or floating-point numbers. Sine and cosine values are inherently in the range **[−1,1]**, and to represent these fractional values in hardware, they need to be scaled into integers using fixed-point arithmetic. By multiplying the fractional values by **256**, they are mapped to the range **[−256,256]**, allowing them to be stored and processed as signed 16-bit integers. This scaling ensures that the fractional precision is preserved while avoiding the complexities of floating-point calculations.resources.

Beam Sweeping:

- The angle increases from 0° to 180° in steps of 1°.
- When the beam reaches 180°, it reverses direction and decreases back to 0°.

Angle Increment Control:

- A counter (angle_counter) increments with every clock pulse (pix_clk).
- Once it matches the ANGLE_INCREMENT_INTERVAL, the angle is updated, ensuring a smooth and adjustable sweep speed.

Pause Functionality:

- When paused is active, angle updates halt, freezing the beam's motion.
- This will also freeze/save the output and position of the object shown in the FPGA.

Distance Measurement

- The ultrasonic sensor measures the time taken for sound waves to return after hitting an object. This time is stored in echo_count.
- The distance in centimeters is calculated using the formula that we mentioned before ***(distance_cm)***

Polar to Cartesian Conversion

- Detected object positions are converted from polar to Cartesian coordinates using:

$$x = r \cos(\theta)$$

$$y = r \sin(\theta)$$

- These calculations align detected objects with the radar beam's position on the VGA display.

VGA Display Control

- The VGA controller generates synchronization signals **(h_sync and v_sync)** and RGB signals to display the radar's output.
- A white beam represents the radar’s scanning motion, while detected objects are displayed as dots or markers.
- The scaling and alignment ensure that the beam and objects are visually synchronized.

Clock Divider

- A clock divider reduces the FPGAs base clock (100 MHz) to a suitable frequency for controlling and sync-ing the VGA display (25 MHz).

Extra Modules used to represent more data:

FPGA Display (seven segments):

- **Digit Multiplexing**: Activates one digit at a time using an output to create a seamless display.
- **Refresh Mechanism**: Uses a 16-bit counter to switch digits at regular intervals for smooth visual updates.
- **Number Decoding**: Extracts hundreds, tens, and units digits from an 8-bit input (`number`) and suppresses leading zeros.
- **Segment Encoding**: Maps digits `0-9` to predefined binary patterns for the seven-segment display.

FPGA Display (led):

- **< 30 cm**: Activates LED pattern `3'b001`, rightmost LED
- **30-39 cm**: Activates LED pattern `3'b010`, middle LED
- **40-49 cm**: Activates LED pattern `3'b100`, leftmost LED
- **≥ 50 cm**: Deactivates all LEDs `3'b000`.