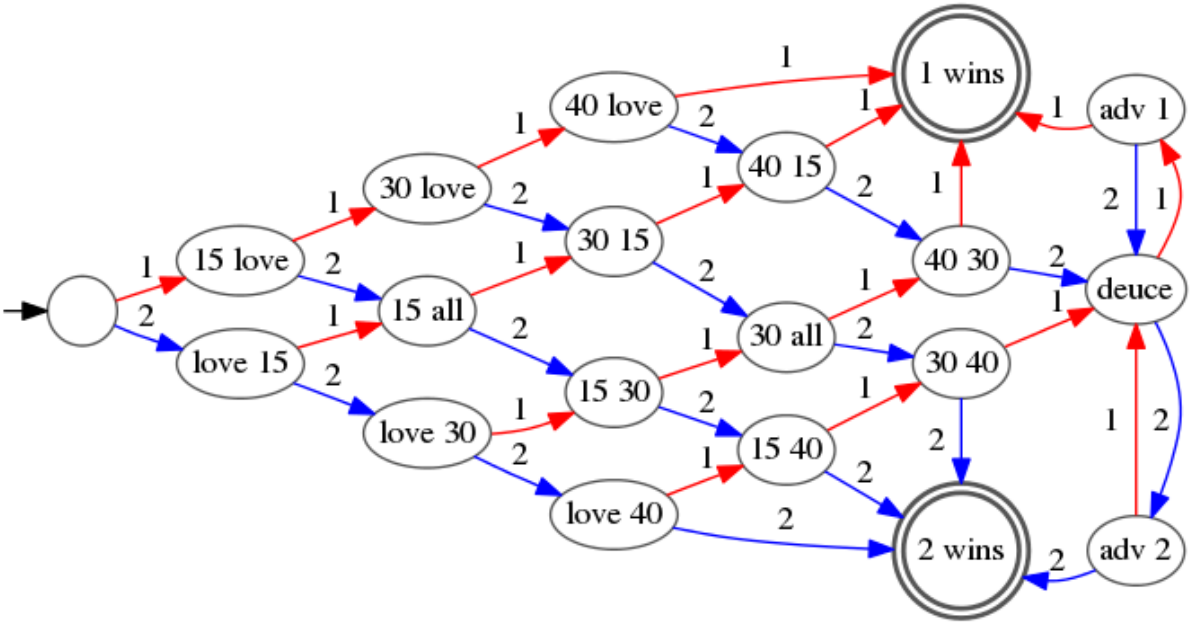
Tennis Scoreboard 🏓

A **complete end-to-end digital design project** that models **official tennis scoring rules** using a **Finite State Machine (FSM)** in **SystemVerilog**, verifies correctness using **assertions and functional coverage**, and visualizes simulation results via a **Python (PyQt5) GUI**.

🧠 Tennis Scoring Logic (FSM)

The design strictly follows official tennis rules:

State	Description
`NORMAL`	Regular scoring (0, 15, 30, 40)
`DEUCE`	Both players at 40
`ADV_P1`	Player 1 advantage
`ADV_P2`	Player 2 advantage
`GAME_P1`	Player 1 wins game
`GAME_P2`	Player 2 wins game



Project Architecture

```
Tennis-Scoreboard-FSM/
│
├── rtl/
│   └── tennis_score_fsm.sv          # FSM RTL (synthesizab
le)
│
├── tb/
│   └── tennis_score_fsm_tb.sv       # Testbench with asser
tions & logging
│
├── simulation/
│   └── modelsim/
│       └── msim_transcript.txt      # Simulation console o
utput
│
├── scripts/
│   └── transcript_to_csv.py         # Transcript → CSV par
ser
│
├── gui/
│   └── tennis_gui.py                # PyQt5 scoreboard GUI
│
├── CSV_of_Scores.csv                # Auto-generated match
data
│
├── README.md
└── LICENSE
```

⚙️ Tools & Technologies

Hardware / Verification

- **SystemVerilog (RTL + TB)**
- **FSM-based design**
- **ModelSim – Intel FPGA Edition**
- **Quartus Prime (RTL compatible)**

Software / Visualization

- **Python 3**
- **PyQt5**
- **CSV parsing**
- **GUI-based scoreboard**

▶️ Steps to Run This Project

🧩 File Overview (Used in Flow)

File	Purpose
`tennis_score_fsm.sv`	RTL FSM for tennis scoring
`tennis_score_fsm_tb.sv`	SystemVerilog testbench
`msim_transcript`	ModelSim simulation output
`Generate_CSV.py`	Transcript → CSV parser
`CSV_of_Scores.csv`	Parsed simulation data
`Tennis_Board_GUI.py`	PyQt5 scoreboard GUI

🧠 Step 1: Compile RTL & Testbench (ModelSim)

Open **ModelSim – Intel FPGA Edition** and navigate to the project directory.

Compile the RTL:

```
vlog -sv tennis_score_fsm.sv
```

Compile the testbench:

```
vlog -sv tennis_score_fsm_tb.sv
```

▶️ Step 2: Run Simulation

Start the simulation:

```
vsim tennis_score_fsm_tb
```

Add signals to waveform:

```
add wave *
```

Run the complete test:

```
run -all
```

📄 Step 3: Locate Simulation Transcript

ModelSim automatically generates a transcript file at:

```
D:\VLSI\Tennis Score Board\simulation\modelsim\msim_transcr
ipt
```

This file contains **time-stamped scoreboard logs**, for example:

```
[T=265000 ns][PHASE=2][DEUCE][P1=40 | P2=40][WIN1=0 WIN2=0]
```

🔄 Step 4: Convert Transcript to CSV

Run the Python script:

```
python Generate_CSV.py
```

This script:

- Reads `msim_transcript`
- Extracts:
  - Time
  - Phase
  - FSM State
  - Player scores
  - Win flags
- Generates:

```
CSV_of_Scores.csv
```

✔️ Output is GUI-ready and spreadsheet-friendly.

📊 Step 5: Verify CSV Output

Example CSV format:

```
Time_ns,Phase,State,P1_Score,P2_Score,WIN1,WIN2
45000,1,NORMAL,0,0,0,0
115000,1,GAME_P1,40,0,1,0
525000,3,GAME_P2,40,40,0,1
```

🖥️ Step 6: Launch Tennis Scoreboard GUI

Run the PyQt5 GUI:

```
python Tennis_Board_GUI.py
```

Step 7: RTL Synthesis Using Intel Quartus Prime

This step validates that the **tennis scoring FSM is synthesizable and FPGA-ready**.

🧱 Step 7.1: Create Quartus Project

1. Open **Intel Quartus Prime**
2. Click **File → New Project Wizard**
3. Set project directory: Example

```
D:\VLSI\Tennis Score Board
```

4. Project name:

```
tennis_score_fsm
```

5. Add files:
   - ✅ `tennis_score_fsm.sv`
   - ❌ Do NOT add testbench

🔌 Step 7.2: Select FPGA Device

Choose device based on your board (example):

```
Family      :CycloneIVE
Device      :EP4CE115F29C7
```

(Device choice does not affect FSM logic correctness)

⚙️ Step 7.3: Configure Top-Level Entity

1. Go to **Assignments → Settings**
2. Under **General**
3. Set **Top-Level Entity**:

```
tennis_score_fsm
```

▶️ Step 7.4: Run RTL Analysis & Synthesis

Click:

```
Processing →Start Compilation
```

Quartus will perform:

- Syntax checking
- FSM extraction
- Logic optimization
- Resource mapping

✅ Step 7.5: Verify Compilation Results

After successful compilation, check:

✔️ Compilation Report

- **No errors**
- FSM inferred correctly

✔️ State Machine Viewer

```
Tools → Netlist Viewers → State Machine Viewer
```

This confirms:

- Normal
- Deuce
- Advantage

- Game Win transitions

📊 Step 7.6: Review Resource Utilization

Open:

```
Compilation Report → Fitter → ResourceSection
```

Typical usage:

- < 50 LUTs
- Minimal registers
- No DSP / RAM blocks

✔️ Confirms **lightweight & efficient design**

📸 Project Snapshots

Refer to the `snapshots/` folder in this repository to visually understand the project.

It contains:

- 🟢 **Simulation waveforms** (ModelSim / Quartus simulation results)
- 🟢 **Quartus GUI screenshots** (project setup, synthesis, compilation)
- 🟢 **FSM diagrams** (state transitions and control logic)

These images help verify functionality, design flow, and implementation clarity without running the project locally.

👤 Author

Vishal Prakash Shinde

LinkedIn: https://www.linkedin.com/in/vishal-shinde-6ab353262/