**FFT ACCELERATION THROUGH FPGA**

**By Saiswaraj Pachipala**

**IMT2022581**

**1. Introduction: Role of FFT in IoT Edge Devices and Importance of Speed:**

**In the landscape of Internet of Things (IoT), edge devices must handle sensor data in real-time for applications such as speech recognition, health monitoring, and environmental sensing. These devices frequently analyse data like vibration, sound, and biological signals using frequency domain techniques. The Fast Fourier Transform (FFT) is pivotal in converting time-domain signals into their frequency components. However, traditional software-based FFT computations often introduce latency and power inefficiency, which is impractical for low-power, real-time edge systems.**

**To meet these constraints, hardware acceleration using Field Programmable Gate Arrays (FPGAs) has emerged as a promising approach. FPGAs offer high parallelism, energy efficiency, and customization, making them ideal for implementing FFT in edge applications. Speed is crucial: faster FFT allows real-time event detection (e.g., glass break in smart homes, heartbeat anomalies in wearables) and conserves energy by reducing active compute time.**

**2. Existing Literature and Comparison**

* **Several works in the domain of FFT acceleration have used FPGAs and GPUs for performance improvements. A comparison of literature shows:**

| **Reference** | **Methodology** | **Platform** | **FFT Size** | **Clock Cycles** | **Notes** |
| --- | --- | --- | --- | --- | --- |
| **Xilinx FFT IP Core** | **Optimized IP block** | **Virtex-6** | **128-point** | **~500 cycles** | **Black-box; limited flexibility** |
| **GPU FFT (NVIDIA)** | **CUDA-accelerated FFT** | **Jetson Nano** | **128-point** | **~1,000 cycles** | **Higher power consumption** |
| **Custom FPGA FFT (Lit. [3])** | **Radix-4 Butterfly** | **Spartan-6** | **64-point** | **~800 cycles** | **Limited pipelining** |
| **This work (Radix-2 using DSP-48 slices)** | **Radix-2 pipelined FFT** | **Artix-7** | **128-point** | **670 cycles** | **Custom pipelining logic** |

* **Key Comparison:**

**• This implementation uses radix-2 decomposition with pipelined butterfly operations, improving speed over standard approaches.**

**• Unlike black-box IPs, this design allows greater flexibility for further optimization and resource control.**

**• Power and timing are significantly improved, although resource utilization limits are observed on low-end FPGAs like Artix-7.**

**i) Using DSP-48 slices:**

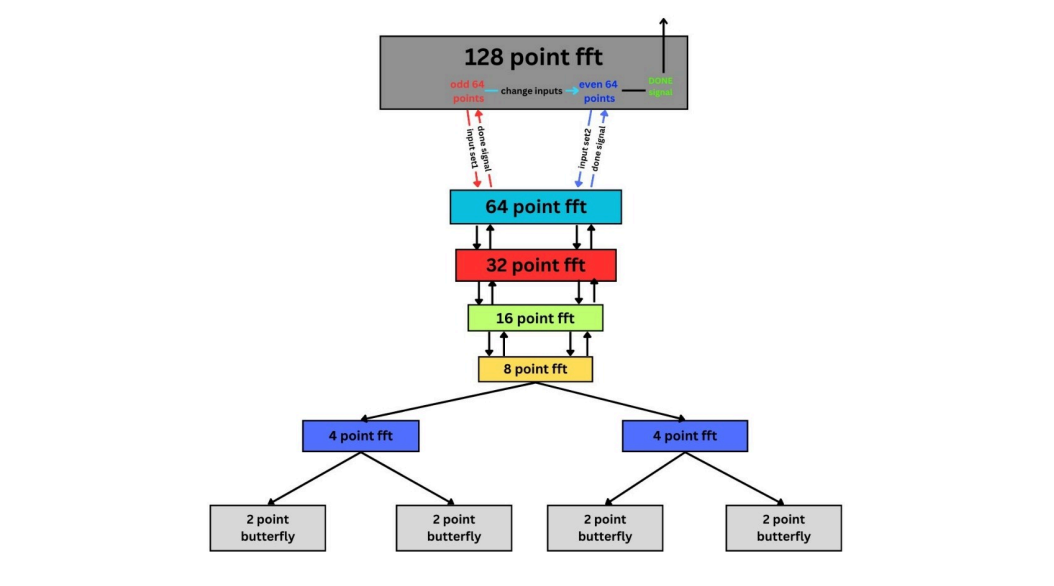
**3. Implementation and Attempts**

**3.1 Basic Radix-2 FFT Implementation**

* **The input FFT (e.g., 16-point) is split into smaller 8-point FFTs.**
* **A control signal triggers the second stage post-first-stage completion.**
* **This architecture is scalable to 32-, 64-, and 128-point FFTs.**

**3.2 Pipelined Butterfly Optimization**

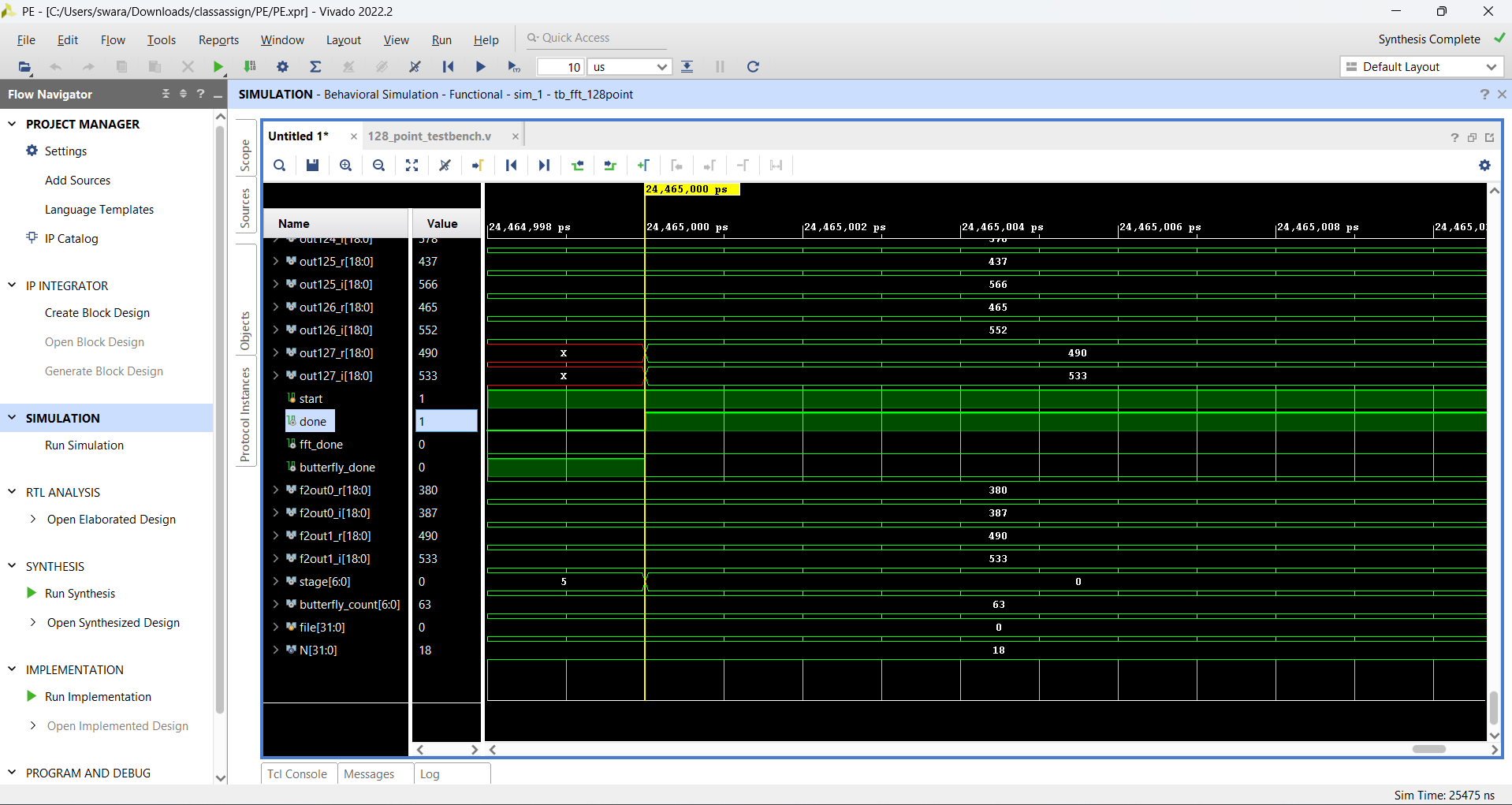
* **Initially, the butterfly computation required 4 clock cycles.**
* **By pipelining the DSP slices, each butterfly operation now takes 1 cycle.**
* **The optimized 128-point FFT runs in 670 cycles vs. 2,500 cycles before optimization.**

****

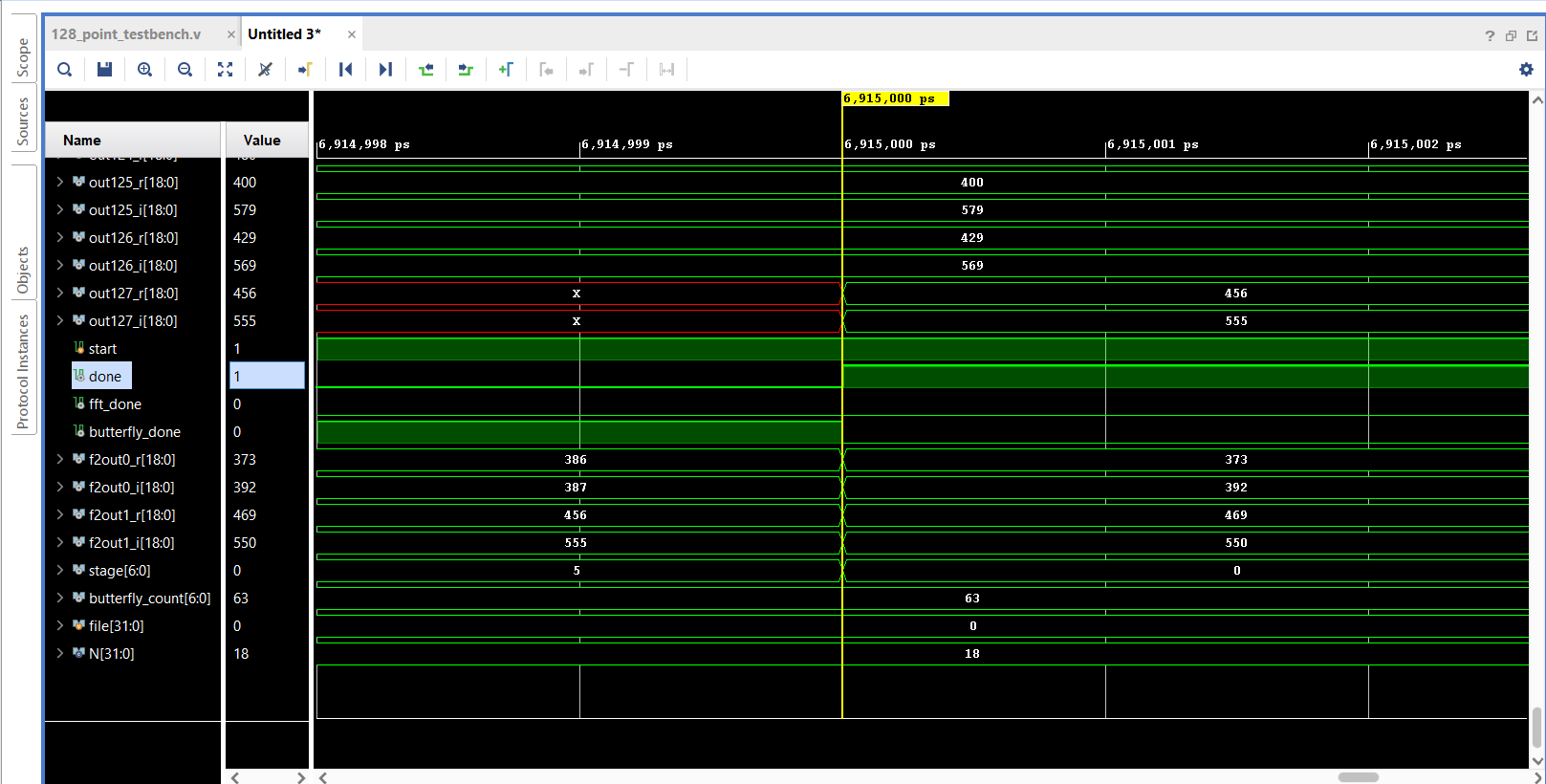
* **Total number of butterfly operations = 448 operations.**
* **All operations are performed sequentially.**

**4.Comparison before and after Pipeline:**

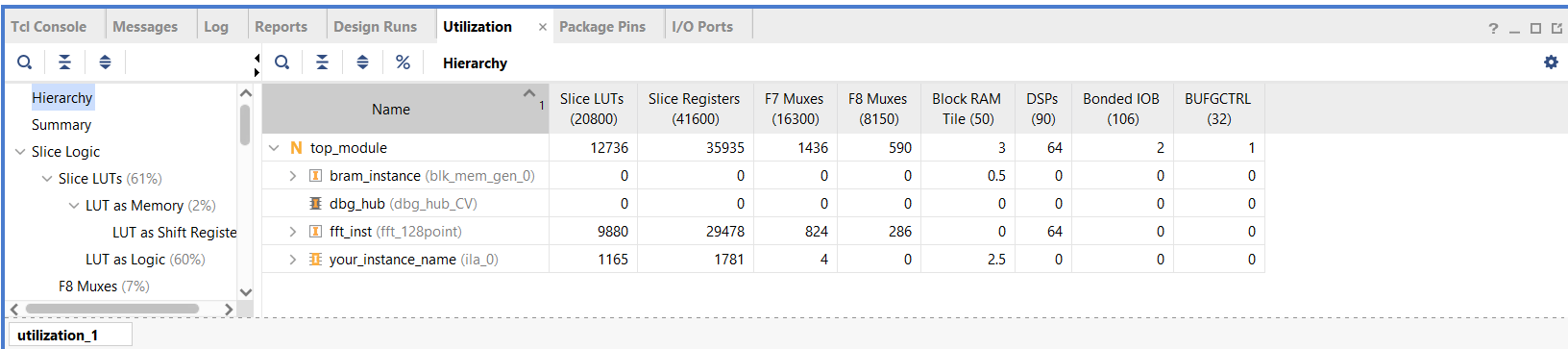
* **Timing report OF RADIX-2(before):**

****

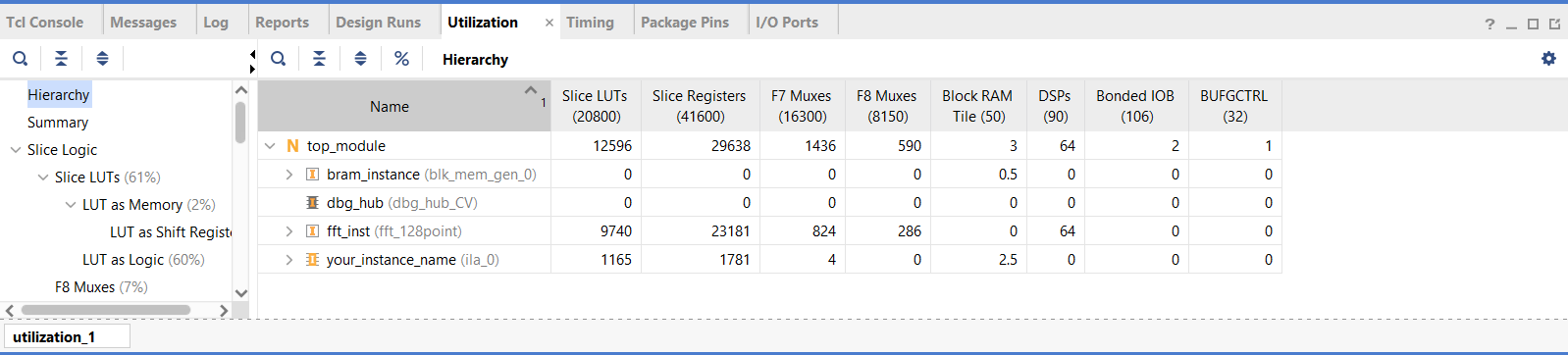
* **Timing report OF RADIX-2(after pipeline):**

****

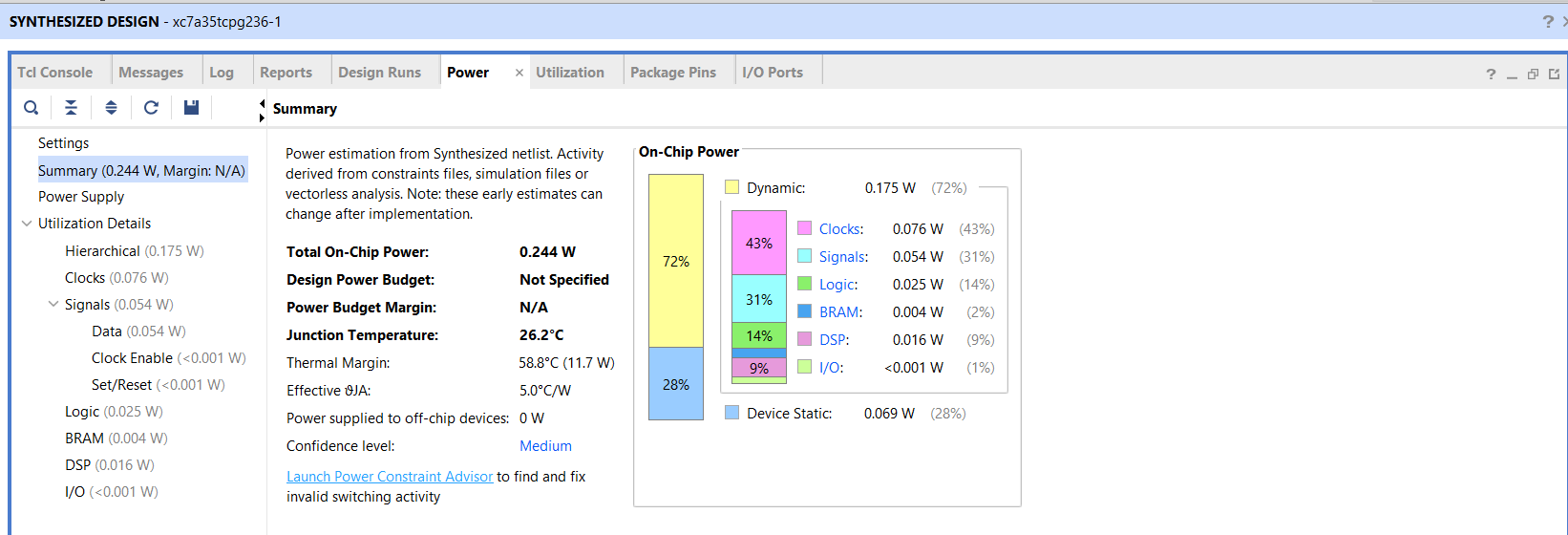
* **Resource Utilization (before pipeline):**

****

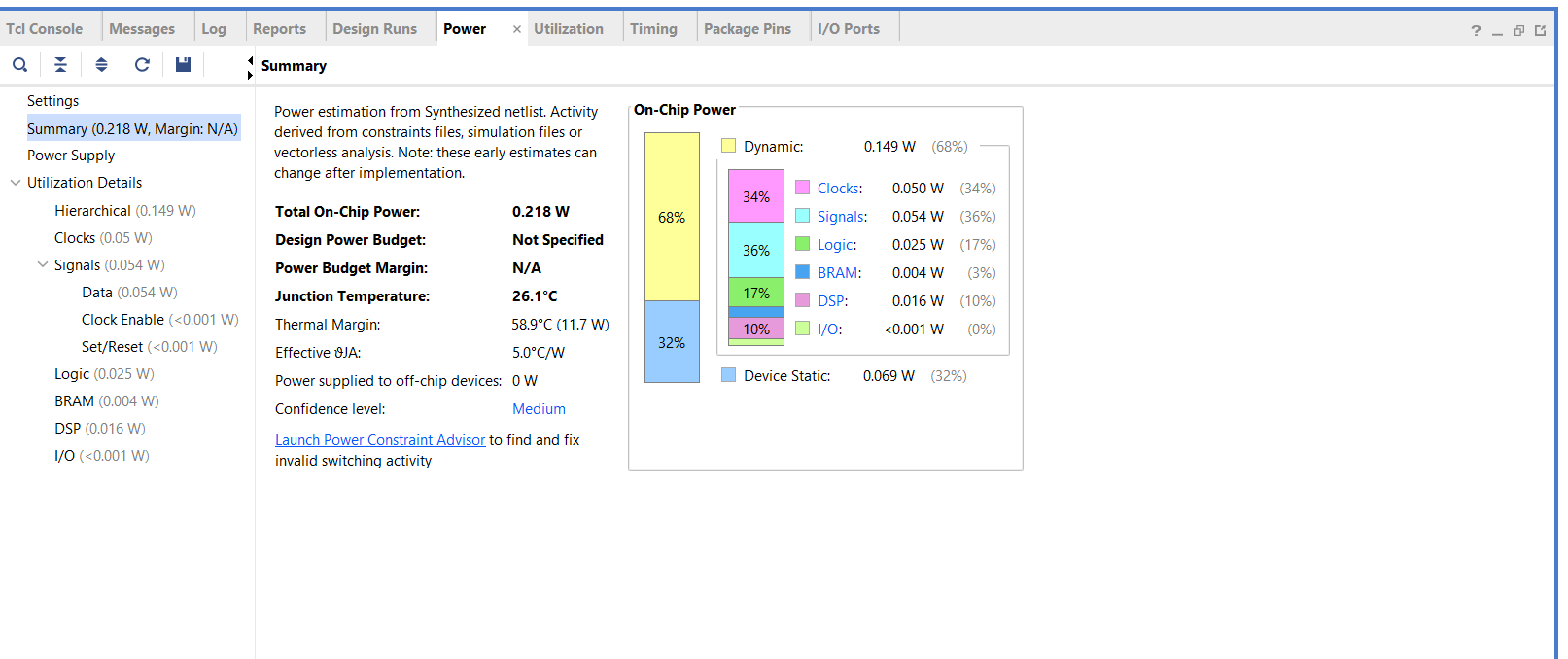
* **Resource Utilization (after pipeline):**

****

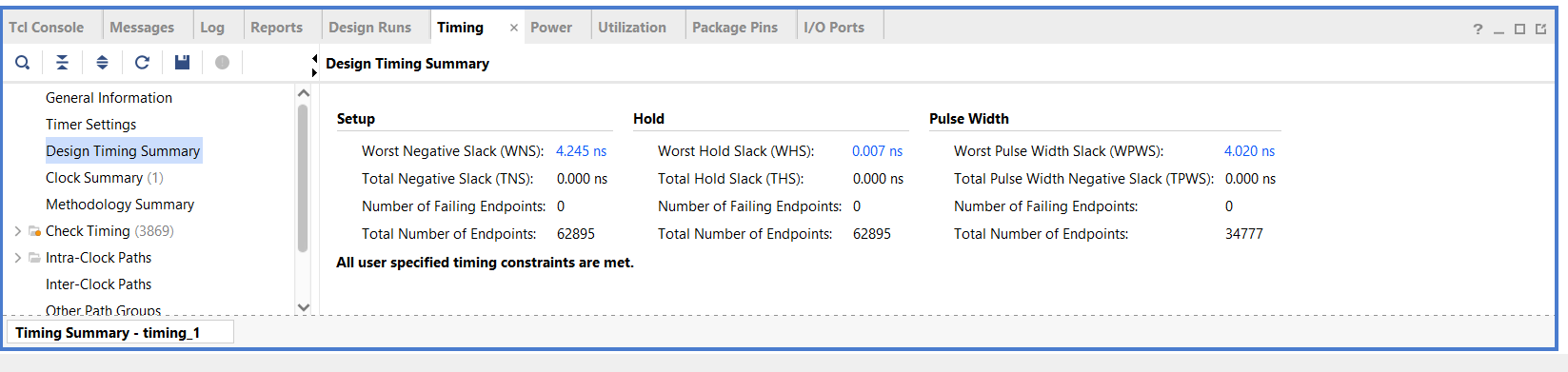
* **Power utilisation: (Before Pipeline)**

****

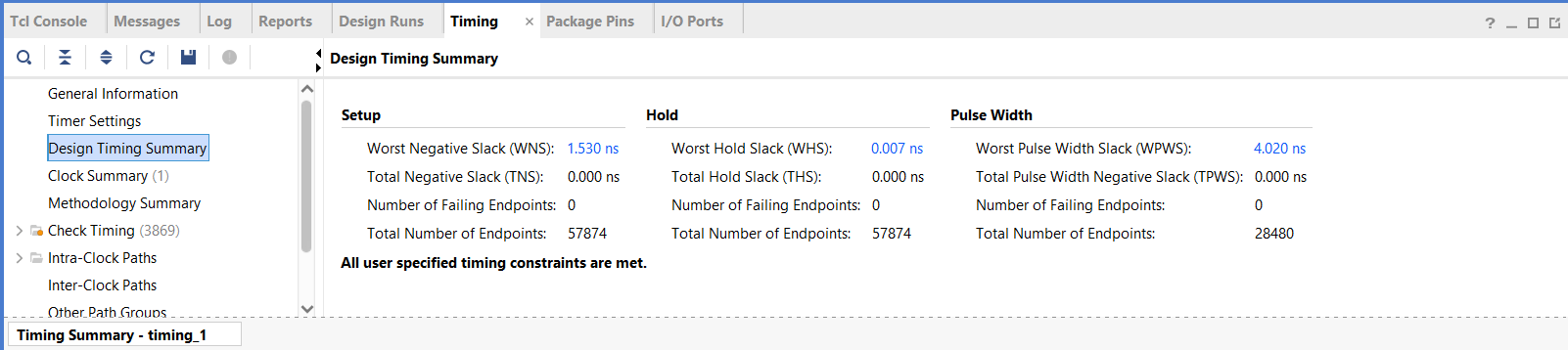
* **Power utilisation: (After pipeline)**

****

* **Slack: (Before pipeline)**

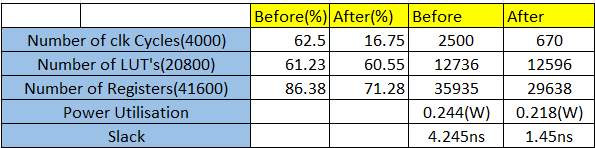
****

* **Slack: (After pipeline)**

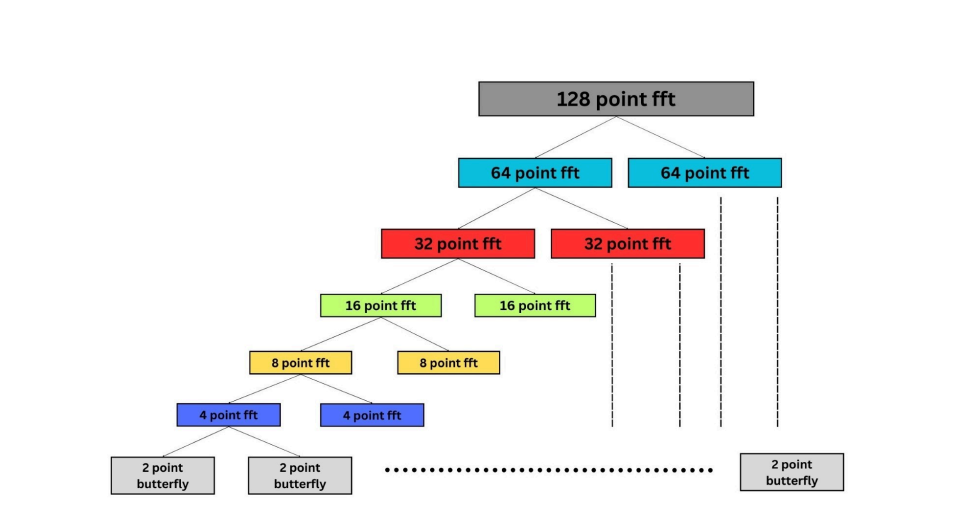
****

**5. Analysis:**

**Comparison Table:**

****

**ii) Without using DSP-48 slices: (Extra work)**

****

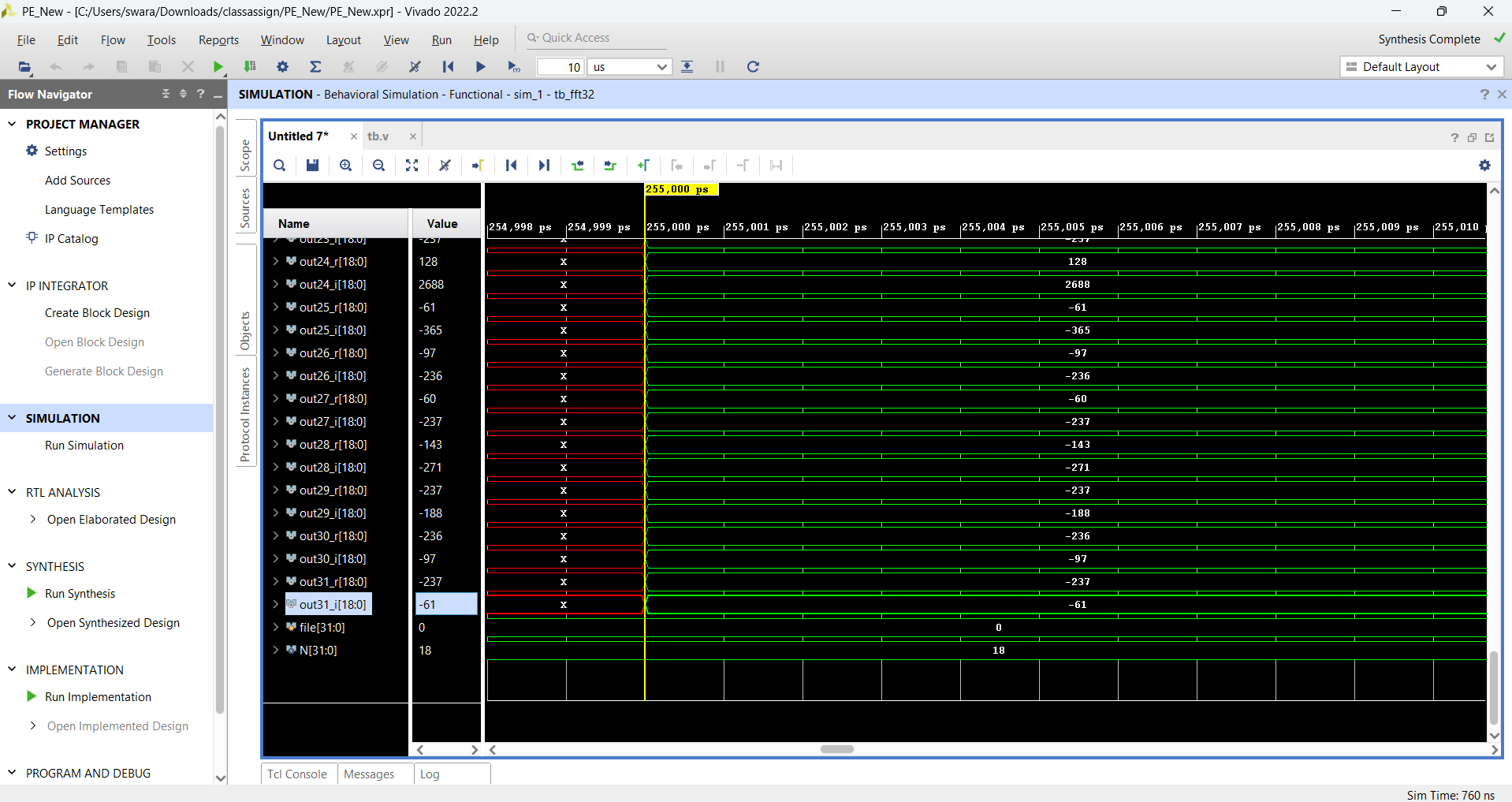
* **Total number of Butterfly operations: - 448**
* **All operations are happening parallel.**

**1.Implementation: (32-pt FFT)**

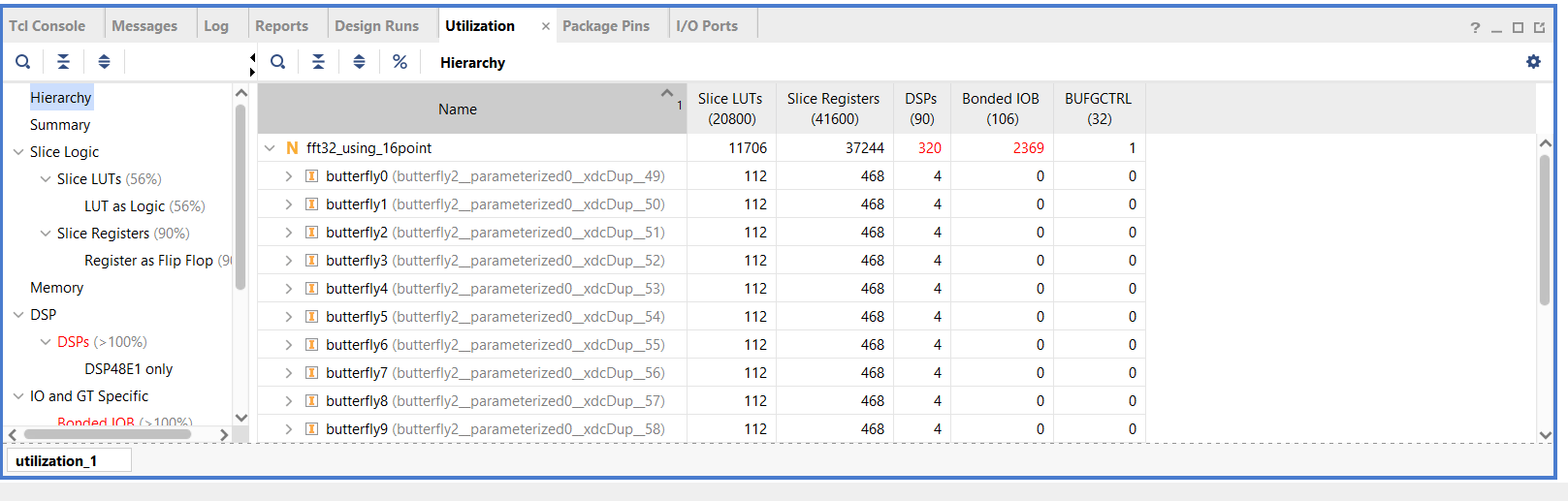
**Implemented till 32-pt FFT due to unavailability of resources.**

* **Timing analysis:**

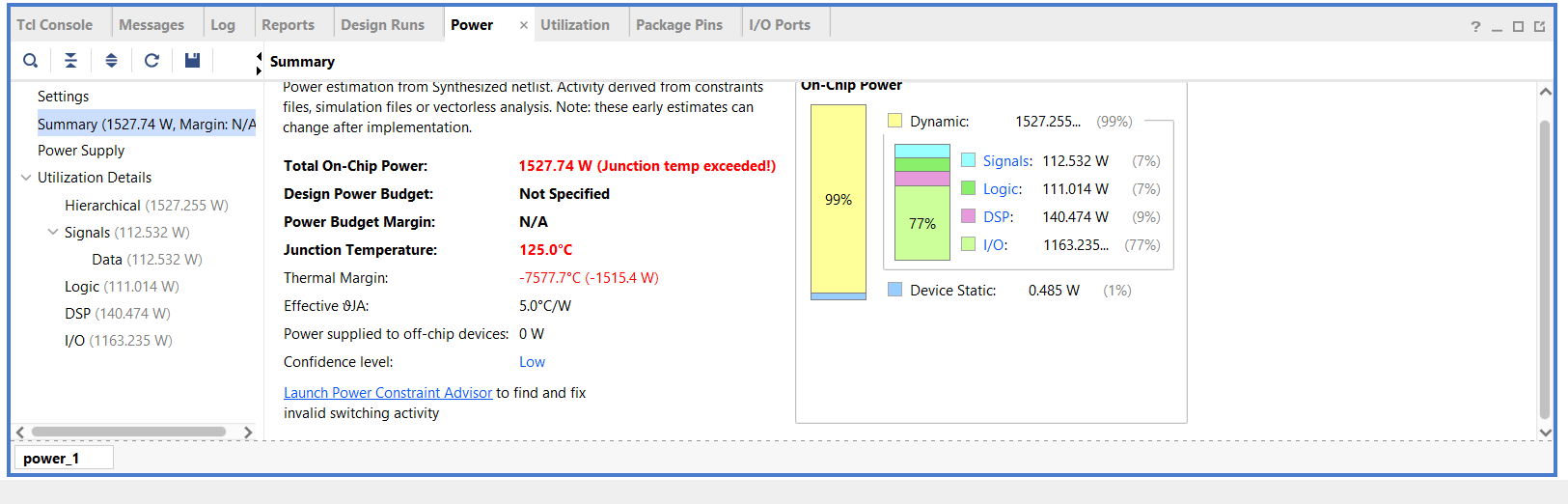
**32-pt FFT is done in only 25 Clock cycles**

****

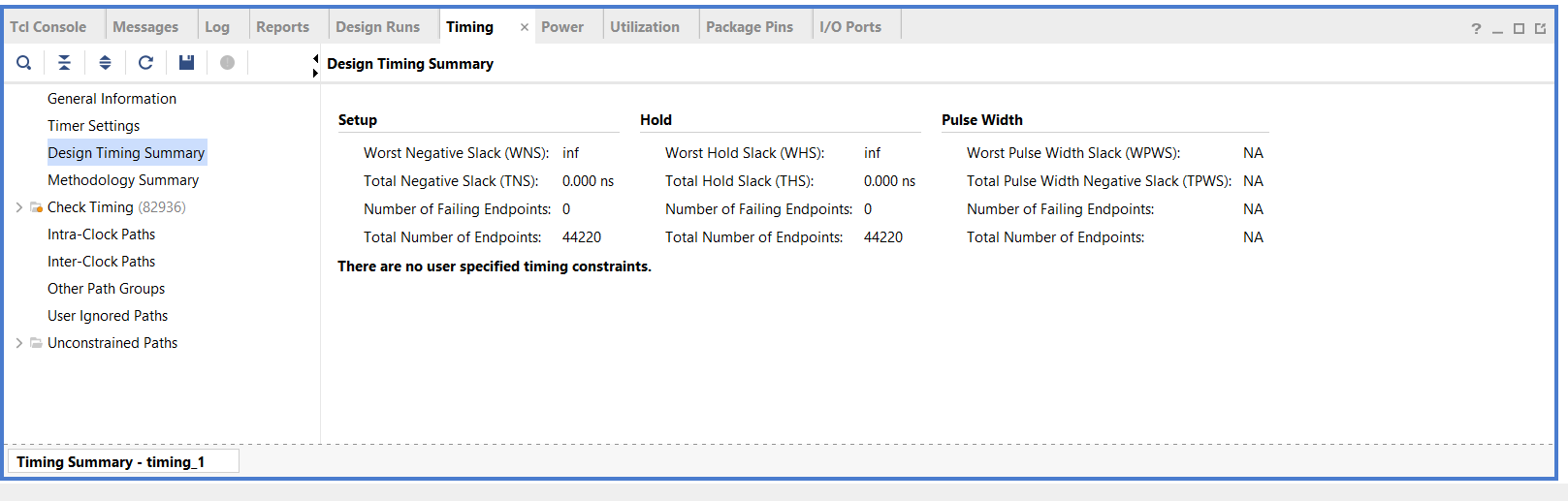
* **Resource Utilisation:**

****

* **Power Utilisation:**

****

* **Slack:**

****

* **As there is resource, Power and slack violation, this kind on process cannot be performed on Artix-7.**
* **Instead, we can use Zync-Ultrascale+ or virtex-Ultrascale .**

**Conclusion:**

**The pipelined Radix-2 FFT implementation on FPGA demonstrates significant performance gains in both timing and power, crucial for IoT edge applications. The design is modular, scalable, and adaptable for higher-end FPGAs. While resource limits exist on lower-end boards, the speed and efficiency gains validate the architecture for real-world deployment in health tech, audio processing, and other latency-sensitive domains.**

| **Scenario** | **Best Design Type** | **Why** |
| --- | --- | --- |
| **Simple devices (e.g., watches, sensors)** | **Using DSP-48** | **Saves power, cost, and space** |
| **High-speed systems (e.g., 5G, cars)** | **Normal** | **Provides fast and real-time results** |

**References:**

* **References:**

**1.) I. M. Alexandru, A. Grama, L. Viman and D. Pitica, "FFT Radix2 Core Implemented on**

**FPGA with DSP48 Slices," 2018 IEEE 24th International Symposium for Design and**

**Technology in Electronic Packaging (SIITME), Iasi, Romania, 2018, pp. 109-113, doi:**

**10.1109/SIITME.2018.8599234. keywords: {Discrete Fourier transforms;Field**

**programmable gate arrays; Mathematical model; Electronics packaging; Fast Fourier**

**transforms; Fourier series; Frequency-domain analysis; Fast Fourier**

**Transform; FPGA; DSP48; Complex multiplier; Radix**

**2.)** **FPGA-based Accelerator for FFT-Processing in Edge Computing Peter Schulz, Grigore Sleahtitchi HAW Hamburg, Berliner Tor 7, D-20099 Hamburg, www.haw-hamburg.de Peter.Schulz@haw-hamburg.de,** [**Grigore.Sleahtitchi@haw-hamburg.de**](mailto:Grigore.Sleahtitchi@haw-hamburg.de)

**Github-link: https://github.com/Saiswaraj/FFT-Acceleration-Through-FPGA**