Power Log, Peak Detection, and Histogram Accumulator IP

Under The Guidance Of :-

Prof. Sankaran Aniruddhan
EEE Department, IIT-MADRAS

Submitted By:-

POTNURU SAMITA
Roll No:- VU22EECE0100371
Gitam University, Andhra Pradesh

REDDY GEETHA SAHITHI
Roll No:- B220495EC
NIT Calicut, Kerala

INDEX:-

1. Problem Statement:- Design an integrated HLS IP core

1. Computes power or log-power per FFT bin
2. Detects peaks that cross a dynamic threshold
3. Accumulates histograms to analyze frequency occupancy over time

2. Workflow:-

1. **Design:** The IP core is implemented in **Vitis HLS C++** (`power.h` and `compute.cpp`). It streams complex FFT samples, computes the power in dBm using a fixed-point log approximation, and compares the result to configurable thresholds:

- **Upper threshold(`max_thresh`)** detects unusually strong signals.
- **Lower threshold(`min_thresh`)** detects signals that drop below a noise floor or expected minimum.

   If the computed power crosses either threshold, the sample is flagged as a peak, and the corresponding frequency bin's histogram counter is incremented.

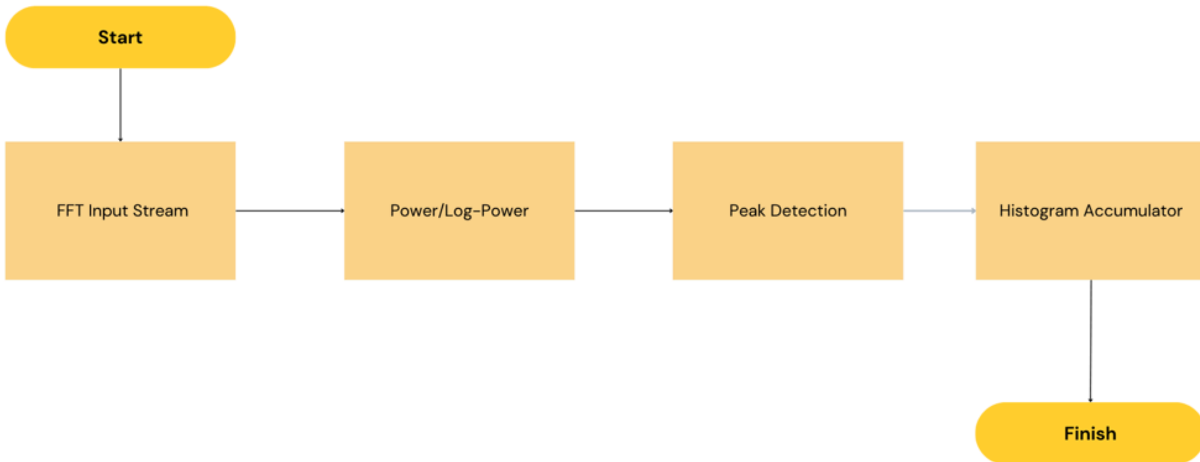
2. **Testbench:**The testbench (`compute_tb.cpp`) verifies the IP core by:

- Reading FFT samples (`real`, `imag`, `bin_index`) from `3input.csv`.
- Resetting the histogram to clear previous counts.
- Streaming all samples through the IP to compute power, detect peaks, and update the histogram.
- Printing debug information for the first few samples in each bin to check the peak detection logic.
- Saving the final histogram to `histogram_output.csv` for visualization.

   The thresholds `min_thresh` and `max_thresh` are configurable and control whether a sample is flagged as a peak. The output verifies that the frequency bins correctly accumulate the number of detected peaks over time.

3. **Visualization:**The histogram output is plotted in Google Colab using Python. The CSV is loaded, frequency bins are labeled in MHz, and a bar graph shows peak counts per bin.The plot verifies how the spectrum is occupied over time.

3.`Flow Chart:-



4. Design Specification:-

Parameter	**Description**
Input Data Format	16-bit signed integer (real & imag)
Throughput	1 sample/cycle (II=1)
Platform	Vitis HLS Software
Pragma	Pipelining & Unrolling

5. Codes:-

1. **Google colab for the input generation in a csv file [1]**

```
import numpy as np
import pandas as pd
from scipy.fft import fft
from math import pi

# === Parameters ===
Fs = 100e6  # Sampling frequency
N = 65536   # Number of samples
num_bins = N // 2  # Positive frequency bins = 32768

# === Generate 65536 random samples using (r, theta) ===
r_min = 0.0219
r_max = 0.7096
```

```
# Random magnitudes and phases
r = np.random.uniform(r_min, r_max, N)
theta = np.random.uniform(0, 2 * pi, N)

# Convert to complex
real_time = r * np.cos(theta)
imag_time = r * np.sin(theta)
samples = real_time + 1j * imag_time

# === Perform FFT ===
fft_result = fft(samples)
fft_positive = fft_result[:num_bins]  # Take only first 32768 positive bins

# === Scale and quantize ===
scale_factor = 2**10  # Scale float to int16 range for HLS
real_fixed = np.round(np.real(fft_positive) * scale_factor).astype(np.int16)
imag_fixed = np.round(np.imag(fft_positive) * scale_factor).astype(np.int16)
fft_bin_number = np.arange(num_bins, dtype=np.uint16)  # 0 to 32767

# === Create CSV dataframe ===
df = pd.DataFrame({
    "real": real_fixed,
    "imag": imag_fixed,
    "fft_bin_number": fft_bin_number
})

# === Save to CSV ===
df.to_csv("fft_input_for_hls.csv", index=False)
print("✅ CSV saved: fft_input_for_hls.csv")

# === Optional: download in Google Colab ===
try:
    from google.colab import files
    files.download("fft_input_for_hls.csv")
except:
    pass
```

2. **power.h**

```
#ifndef POWER_H
#define POWER_H
```

```
#include <ap_int.h>
#include <ap_fixed.h>
#include <hls_stream.h>

#define N_SAMPLES 32768
#define N_BINS 16

#define  BIN_FREQ_RES  1525                    // ≈ 100MHz / 65536 (FFT bin
spacing)
#define MAX_FREQ      50000000            // Nyquist frequency: 50 MHz
#define FREQ_BIN_WIDTH (MAX_FREQ / N_BINS) // Each bin ≈ 3.125 MHz

// Type definitions
typedef ap_int<16>    data_t;
typedef ap_uint<15>   bin_index_t;
typedef ap_uint<4>    bin_t;
typedef ap_uint<10>   hist_t;
typedef ap_fixed<16, 8> dbm_t;  // Q8.8 fixed-point type for dBm

// Input sample format
struct sample_t {
   data_t real;
   data_t imag;
   bin_index_t bin_index;
};

// Function to approximate 10*log10(power) using fixed-point math
dbm_t log10_approx(ap_uint<32> power);

// Top-level IP function for power computation and histogram update
void geeth_power(
   hls::stream<sample_t>& in_stream,
   hist_t histogram[N_BINS],
   bool reset_histogram,
   dbm_t min_thresh,
   dbm_t max_thresh
);

#endif
```

3. **compute.cpp**

```
#include "power.h"

// Accurate fixed-point 10*log10(x) approximation using log2(x)
// Q8.8 format → multiply by 256
dbm_t log10_approx(ap_uint<32> power) {
#pragma HLS inline

   // Avoid log(0)
   if (power == 0) power = 1;

   // Count leading zeros to estimate log2
   ap_uint<6> lz = __builtin_clz(power);
   ap_uint<6> log2_int = 31 - lz;

   // Normalize: shift power left so MSB is at bit 31
   ap_uint<32> norm = power << lz;

   // Extract fractional part from high bits
   ap_uint<8> frac = (norm >> 23) & 0xFF; // Top 8 bits after MSB

   // log2_fixed = int_part << 8 + frac (Q8.8)
   ap_uint<16> log2_fixed = (log2_int << 8) | frac;

   // log10(x) ≈ log2(x) * log10(2) → log10(2) ≈ 0.30103
   // So 10*log10(x) ≈ log2(x) * 10 * 0.30103 ≈ log2(x) * 77 in Q8.8
   dbm_t result = (log2_fixed * 77) >> 8;
   return result;
}

void geeth_power(
   hls::stream<sample_t>& in_stream,
   hist_t histogram[N_BINS],
   bool reset_histogram,
   dbm_t min_thresh,
   dbm_t max_thresh
) {
#pragma HLS INTERFACE axis port=in_stream
```

```
#pragma HLS INTERFACE s_axilite port=histogram bundle=CTRL
#pragma HLS INTERFACE s_axilite port=reset_histogram bundle=CTRL
#pragma HLS INTERFACE s_axilite port=min_thresh bundle=CTRL
#pragma HLS INTERFACE s_axilite port=max_thresh bundle=CTRL
#pragma HLS INTERFACE s_axilite port=return bundle=CTRL
#pragma HLS PIPELINE II=1

   static hist_t hist_local[N_BINS] = {0};
#pragma HLS ARRAY_PARTITION variable=hist_local complete dim=1

   if (reset_histogram) {
       for (int i = 0; i < N_BINS; i++) {
#pragma HLS UNROLL
           hist_local[i] = 0;
       }
   }

   // Read one sample per call
   sample_t s = in_stream.read();

   // Shift input to remove LSB noise
   data_t a = s.real >> 6;
   data_t b = s.imag >> 6;

   // Power = a^2 + b^2
   ap_uint<32> power = a * a + b * b;

   // Compute 10*log10(power) in Q8.8
   dbm_t dbm = log10_approx(power);

   // Determine if it's a peak
   bool peak = (dbm > max_thresh || dbm < min_thresh);

   // Get frequency bin from FFT index
   int freq_hz = s.bin_index * BIN_FREQ_RES;
   bin_t bin = freq_hz / FREQ_BIN_WIDTH;
   if (bin >= N_BINS) bin = N_BINS - 1;

   // Accumulate peak count
   if (peak && !reset_histogram) {
```

```
        hist_local[bin]++;
    }

    // Update output histogram
    for (int i = 0; i < N_BINS; i++) {
#pragma HLS UNROLL
        histogram[i] = hist_local[i];
    }
}
```

4. **compute_tb.cpp**

```
#include "power.h"
#include <fstream>
#include <iostream>
#include <sstream>
#include <string>

int main() {
    hls::stream<sample_t> in_stream;
    hist_t histogram[N_BINS] = {0};
    int debug_count[N_BINS] = {0};

    std::ifstream infile("3input.csv");
    if (!infile.is_open()) {
        std::cerr << "❌ ERROR: Could not open 3input.csv\n";
        return 1;
    }

    std::string line;
    getline(infile, line); // skip header

    const dbm_t min_thresh = -30;
    const dbm_t max_thresh = 20;

    // Step 1: Reset histogram
    geeth_power(in_stream, histogram, true, min_thresh, max_thresh);

    // Step 2: Stream all input samples
```

```
    while (getline(infile, line)) {
        sample_t s;
                  sscanf(line.c_str(),   "%hd,%hd,%hu",   &s.real,   &s.imag,
&s.bin_index);
        in_stream.write(s);
        geeth_power(in_stream, histogram, false, min_thresh, max_thresh);
    }
    infile.close();

    // Step 3: Print debug info (first 5 samples per bin)
    std::ifstream infile_debug("3input.csv");
    getline(infile_debug, line); // skip header again

    std::cout << "\n🔍 Debug Info (First 5 Samples Per Bin):\n";

    while (getline(infile_debug, line)) {
        sample_t s;
                  sscanf(line.c_str(),   "%hd,%hd,%hu",   &s.real,   &s.imag,
&s.bin_index);

        data_t a = s.real >> 6;
        data_t b = s.imag >> 6;
        ap_uint<32> power = a * a + b * b;

        dbm_t dbm = log10_approx(power);
        bool peak = (dbm > max_thresh || dbm < min_thresh);

        int freq_hz = s.bin_index * BIN_FREQ_RES;
        bin_t bin = freq_hz / FREQ_BIN_WIDTH;
        if (bin >= N_BINS) bin = N_BINS - 1;

        if (debug_count[bin] < 5) {
            std::cout << "  Bin " << (int)bin
                      << " | real = " << s.real
                      << ", imag = " << s.imag
                      << ", power = " << (float)dbm
                      << " dBm, peak = " << (peak ? "YES" : "NO") << "\n";
            debug_count[bin]++;
        }
    }
```

```
    infile_debug.close();

    // Step 4: Save histogram to CSV
    std::ofstream out_csv("histogram_output.csv");
    if (!out_csv.is_open()) {
        std::cerr << "❌ ERROR: Could not write to histogram_output.csv\n";
        return 1;
    }

    out_csv << "Bin,Freq_Start_Hz,Freq_End_Hz,Peak_Count\n";
    std::cout << "\n📊 Final Peak Histogram (Frequency Bins):\n";

    for (int i = 0; i < N_BINS; i++) {
        int f_start = i * FREQ_BIN_WIDTH;
        int f_end = (i + 1) * FREQ_BIN_WIDTH - 1;

         std::cout << "  Bin " << i << " [" << f_start << " Hz – " << f_end
<< " Hz] → Peaks: "
                  << histogram[i] << "\n";

            out_csv << i << "," << f_start << "," << f_end << "," <<
histogram[i] << "\n";
    }

    out_csv.close();
        std::cout << "\n✅ histogram_output.csv has been saved
successfully!\n";

    return 0;
}
```

5. **Google colab for histogram plotting**

```
import pandas as pd
import matplotlib.pyplot as plt

# Read histogram CSV
```

```python
df = pd.read_csv("histogram_output.csv")

# Convert frequency to MHz with decimal precision
df["Freq_Range"] = df.apply(
    lambda row: f"{row['Freq_Start_Hz'] / 1e6:.3f}-{row['Freq_End_Hz'] /
1e6:.3f} MHz", axis=1
)

# Plot
plt.figure(figsize=(14, 6))
plt.bar(df["Freq_Range"],      df["Peak_Count"],      color='steelblue',
edgecolor='black')
plt.xticks(rotation=45, ha='right')
plt.xlabel("Frequency Range (MHz)")
plt.ylabel("Peak Count")
plt.title("Histogram of Peaks Detected per Frequency Bin")
plt.grid(axis='y', linestyle='--', alpha=0.6)
plt.tight_layout()
plt.savefig("histogram_plot_mhz.png")
plt.show()
```

6. Outputs

Bin,Freq_Start_Hz,Freq_End_Hz,Peak_Count
0,0,3124999,285
1,3125000,6249999,332
2,6250000,9374999,315
3,9375000,12499999,342
4,12500000,15624999,321
5,15625000,18749999,322
6,18750000,21874999,351
7,21875000,24999999,312
8,25000000,28124999,341
9,28125000,31249999,352
10,31250000,34374999,322
11,34375000,37499999,333
12,37500000,40624999,298
13,40625000,43749999,376
14,43750000,46874999,319
15,46875000,49999999,314

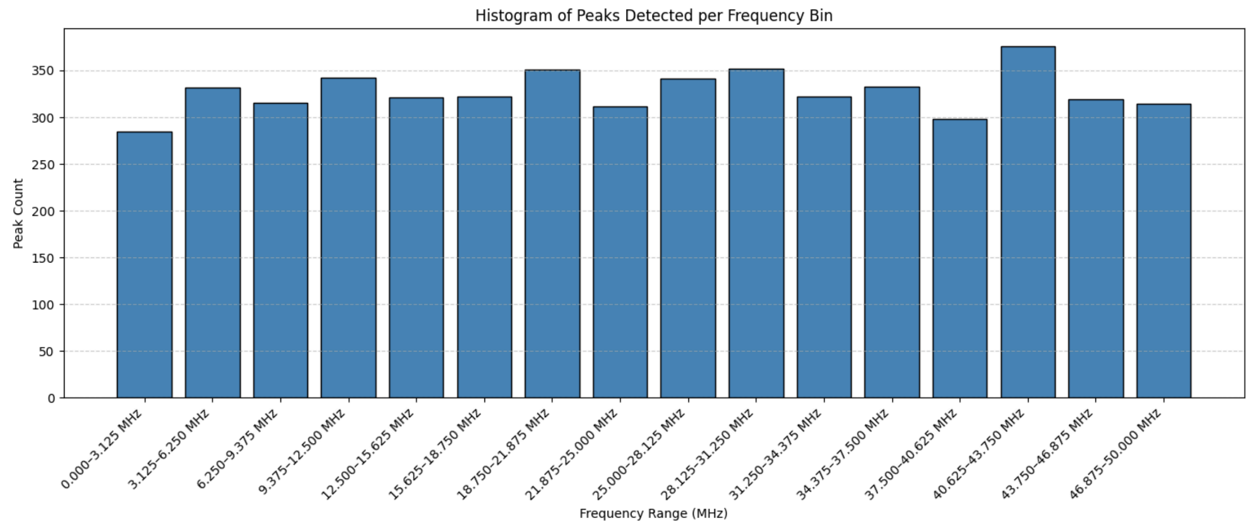


Figure 1: Frequency Range vs Histogram

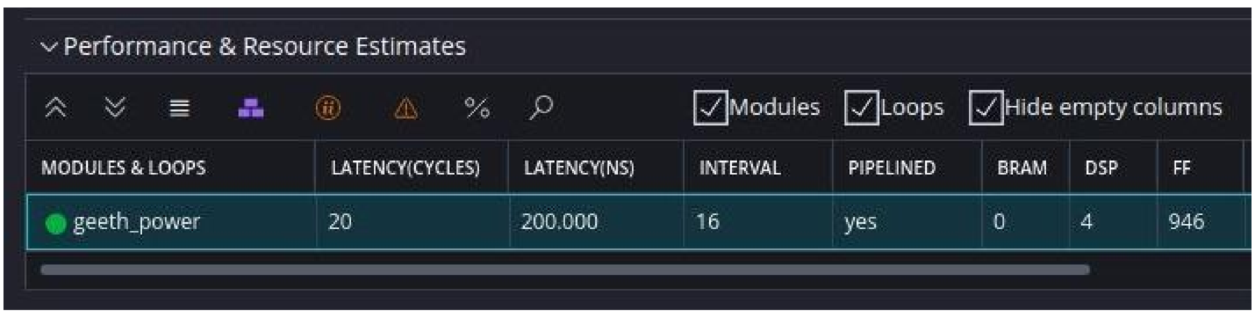


Figure 2 : Performance of the Code

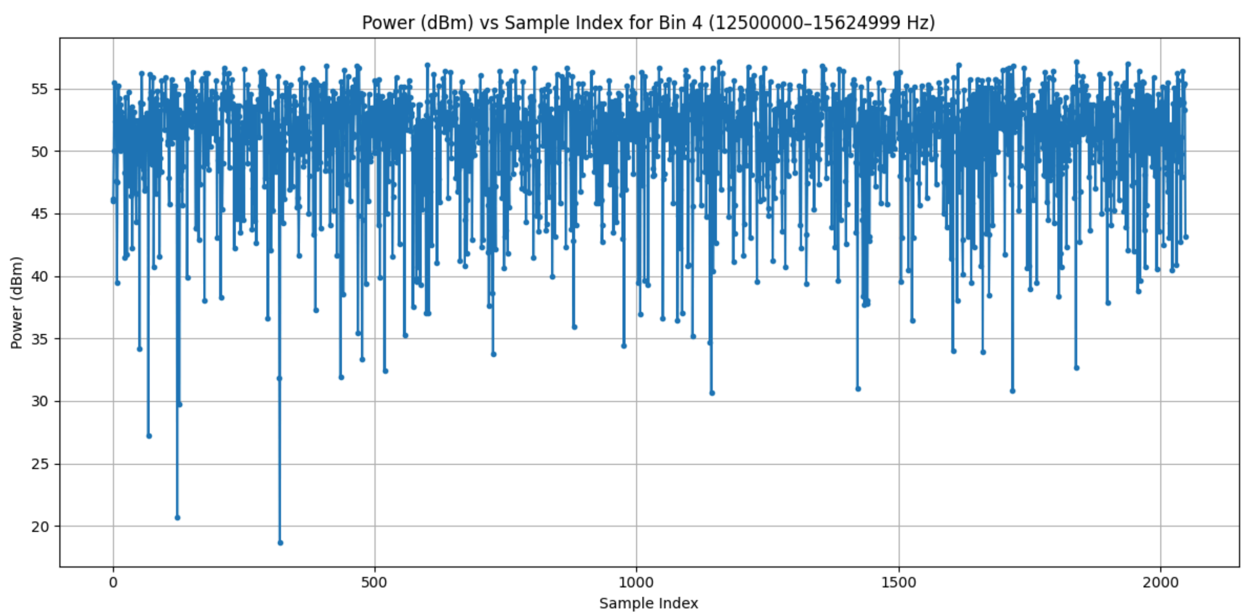


Figure 3 : Sample index vs power (dbm)

6. Challenges Faced:-

- Debugging stream interfaces and ensuring `II=1` pipelining.
- Designing an accurate fixed-point `log10` approximation[2].

7. Conclusion:-

In this project, an integrated Power Log, Peak Detection, and Histogram Accumulator IP was successfully designed, simulated, and verified. FFT samples were processed in real-time to compute power or log-power, detect peaks above or below dynamic thresholds, and accumulate frequency bin histograms.

8. Weekly Report:-

Week 1&2: Learned basics of RTL and HDL Bits
Week 3: Given Problem Statement[3]
Week 4: Did the functionality of the problem statement in vitis[4]
Week 5: Input Generation in Google Colab[5]
Week 6: Integrating the Google Colab and Vitis to get the histogram output

9. References:-

1. https://docs.google.com/spreadsheets/d/1tYSwBJcjGEta9Bw-4-LUpVkdr5pXFV5g0q6ZWoetrOA/edit?usp=sharing
2. https://coretechgroup.com/dbm-calculator/
3. https://www.xilinx.com/support/documents/sw_manuals/xilinx2022_2/ug1399-vitis-hls.pdf
4. https://www.youtube.com/@nitinchandrachoodan6783/playlists
5. https://www.youtube.com/watch?v=GlbJxX130iE