Synthesis of 32 point Fast Fourier Transform FFT

- The synthesis was done using Vitis 2023.2 installed locally. Solutions with different optimization directives were created and compared using the GUI interface.

Case 1: Recursive FFT

- A divide and conquer approach was used to implement the FFT in a recursive manner, by calculating the FFTs of even elements and odd separately and combining them with appropriate twiddle factors.

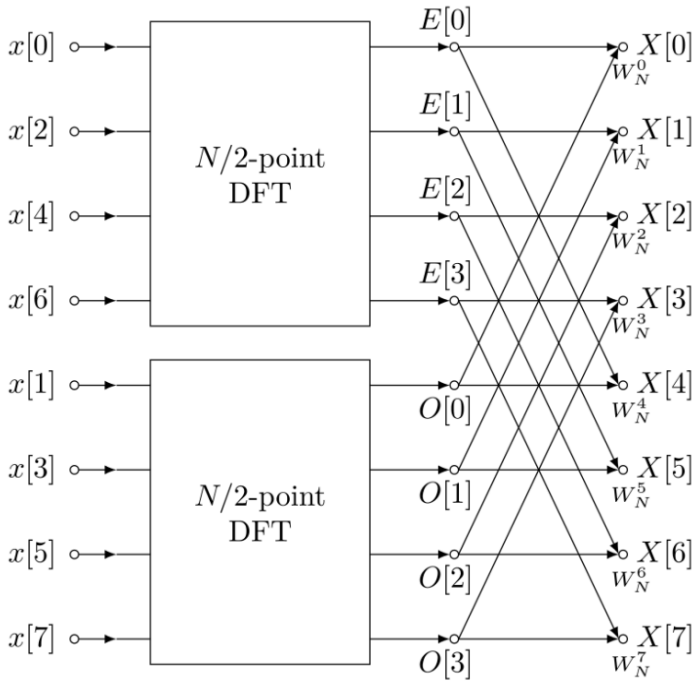

Fig 1. Recursive FFT structure

- Upon synthesis of the FFT_base recursive code, the following error was encountered:

Synthesis	
ERROR: [HLS 200-1715] Encountered problem during source synthesis	fft_base:solution1
ERROR: [HLS 214-139] Recursive function calls are not supported: fft(std::complex<float>*, std::complex<float>*,	fft_base:solution1

- This indicated that recursively building modules was not an accurate way of synthesis.

```
void fft(data_comp in[], data_comp out[], int M){

    if (M==1){
        out[0] = in[0];
        return;
    }

    data_comp x_even[M/2], x_odd[M/2], F_even[M/2], F_odd[M/2];

    for (int i=0;i<M/2;i++){
        x_even[i] = in[2*i];
        x_odd[i] = in[2*i+1];
    }

    fft(x_even,F_even,M/2);
    fft(x_odd,F_odd,M/2);

    for (int k=0;k<M/2;k++){
        data_comp w = data_comp(cos(2*pi*k/M),-sin(2*pi*k/M)) ;
        out[k] = F_even[k] + w*F_odd[k];
        out[k + M/2] = F_even[k] - w*F_odd[k];
    }
}

void FFT(data_comp data_IN[N], data_comp data_OUT[N]){
    fft(data_IN,data_OUT,N);
}
```

Fig 3. Recursive FFT code

Case 2: Non recursive FFT code

- The FFT was implemented in a non-recursive way by unrolling the nested FFT computations and reordering the bits to be input to the first stage of the LOG(N) stages.

```
for (int len = 2; len <= n; len <<= 1) {

    data_comp wlen(cos(2 * pi / len), sin(2 * pi / len));

    for (int i = 0; i < n; i += len) {
        data_comp w(1);

        for (int j = 0; j < len / 2; j++) {
            data_comp u = b[i+j], v = b[i+j+len/2] * w;
            b[i+j] = u + v;          // in place DFT op
            b[i+j+len/2] = u - v;
            w *= wlen;
        }
    }
}
```

- The design was successfully synthesized and the following results were obtained:

Timing Estimate

Target	Estimated	Uncertainty
10.00 ns	7.404 ns	2.70 ns

Performance & Resource Estimates

Modules Loops

Modules & Loops	Issue Type	Violation Type	Distance	Slack	Latency(cycles)	Latency(ns)	Iteration Latency	Interval	Trip Count	Pipelined	BRAM	DSP	FF	LUT	URAM
FFT				-0.10	-	-	-	-	-	no	5	83	11218	17667	0
FFT_Pipeline_VITIS_LOOP_28_1				-	35	350.000	-	35	-	no	0	0	28	69	0
VITIS_LOOP_33_2	II&Timing Violation			-	-	-	-	-	5	no	-	-	-	-	-

Resource utilization:

Utilization Estimates

Summary

Name	BRAM_18K	DSP	FF	LUT	URAM
DSP	-	-	-	-	-
Expression	-	-	0	1381	-
FIFO	-	-	-	-	-
Instance	5	83	10093	15334	-
Memory	-	-	-	-	-
Multiplexer	-	-	-	952	-
Register	-	-	1125	-	-
Total	5	83	11218	17667	0
Available	280	220	106400	53200	0
Utilization (%)	1	37	10	33	0

Case 3: Non recursive FFT code pipelined

- The pipeline directive was applied to the non recursive function that iterates over all the LOGN stages and the following metrics were observed.

Timing Estimate

Target	Estimated	Uncertainty
10.00 ns	7.979 ns	2.00 ns

Performance & Resource Estimates

Modules Loops

Modules & Loops	Issue Type	Violation Type	Distance	Slack	Latency(cycles)	Latency(ns)	Iteration Latency	Interval	Trip Count	Pipelined	BRAM	DSP	FF	LUT	URAM
FFT				-	-	-	-	-	-	no	0	50	11223	12385	0
FFT_Pipeline_FFT_label_bit_reversal				-	35	350.000	-	35	-	no	0	0	28	69	0
FFT_label1				-	-	-	-	-	5	no	-	-	-	-	-

- Unrolling the inner loops while pipelining the outer one should result in an increase in hardware usage but that is not reflected here. I hope to work out the intricacies of pragma and directive applications in Vitis and update this report with better cases in the future.