/\*

\* Supplementary C: Source Code of FFT Algorithm in C language

\*

\* Author: Dr. Ehsan Ali

\* Email: ehssan.aali@gmail.com

\* Last Update: 4th Nov. 2024

\*

\* Used in Article: Innovative Hardware Accelerator Architecture

\* for FPGA-Based General-Purpose RISC Microprocessors

\*

\* URL: https://github.com/ehsan-ali-th/cortex\_m0\_MA/tree/master/Supplementaries

\* \*/

#include <math.h>

#include <stdio.h>

#include <stdlib.h>

#define points 8 /\* for 2^8 = 256 points \*/

#define N (1<<points) /\* N-point FFT \*/

typedef float real;

typedef struct {

real Re;

real Im;

} complex;

#ifndef PI

# define PI 3.14159265358979323846264338327950288

#endif

void fft (complex \*wave, int n, complex \*tmp ) {

if(n>1) { /\* return if n =< 0 \*/

int k,m; complex z, w, \*vo, \*ve;

ve = tmp; vo = tmp+n/2;

for(k=0; k<n/2; k++) {

ve[k] = wave[2\*k];

vo[k] = wave[2\*k+1];

}

fft (ve, n/2, wave); // FFT on even-indexed elements of wave[]

fft (vo, n/2, wave); // FFT on odd-indexed elements of wave[]

for (m=0; m<n/2; m++) {

w.Re = cos(2\*PI\*m/(double)n);

w.Im = -sin(2\*PI\*m/(double)n);

z.Re = w.Re\*vo[m].Re - w.Im\*vo[m].Im; /\* Re(w\*vo[m]) \*/

z.Im = w.Re\*vo[m].Im + w.Im\*vo[m].Re; /\* Im(w\*vo[m]) \*/

wave[m].Re = ve[m].Re + z.Re;

wave[m].Im = ve[m].Im + z.Im;

wave[m+n/2].Re = ve[m].Re - z.Re;

wave[m+n/2].Im = ve[m].Im - z.Im;

}

}

return;

}

// Program entry point.

int main() {

complex wave[N], scratch[N];

int k;

/\* Fill wave[] with a sine wave of known frequency \*/

for (k=0; k<N; k++) {

wave[k].Re = 0.125\*cos(2\*PI\*k/(double)N);

wave[k].Im = 0.125\*sin(2\*PI\*k/(double)N);

}

fft (wave, N, scratch); // Perform FFT, wave[] will have the result.

return 0;

}