Министерство науки и высшего образования РФ

Федеральное государственное бюджетное

образовательное учреждение высшего образования

ВЯТСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

Факультет автоматики и вычислительной техники

Кафедра радиоэлектронных средств

Отчет по лабораторной работе № 1

**Реализация КИХ-фильтра**

дисциплина «Проектирование устройств и систем на цифровых сигнальных процессорах»

|  |  |
| --- | --- |
| Выполнили: студенты группы ИНБс-5301 | \_\_\_\_\_\_\_\_\_\_\_\_\_\_/А.А. Носкова/ |
|  |  |
| Проверил: старший преподаватель кафедры РЭС | \_\_\_\_\_\_\_\_\_\_\_\_\_\_/М.А. Земцов/ |

Киров 2023

**Цель работы:** исследование основных типов весовых функций (окон);

изучение средств визуализации данных Code Composer Studio.

**1) Изучение прямоугольного окна**

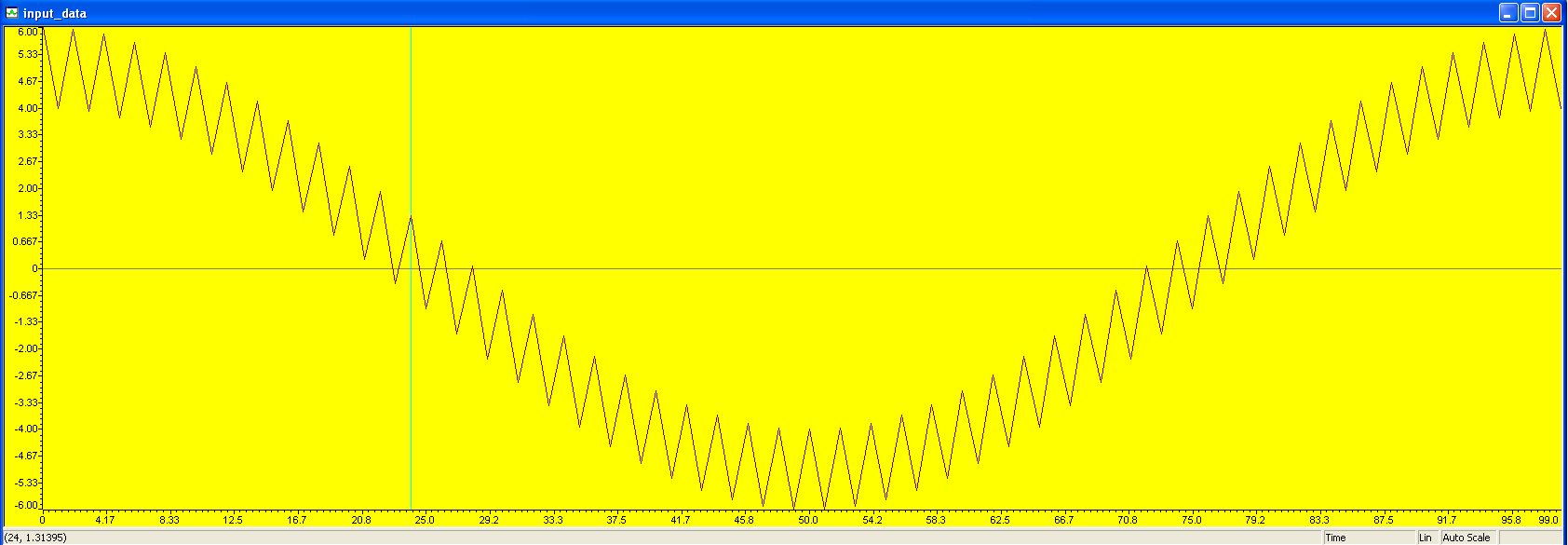


Рисунок 1 – Входные данные (временная область)

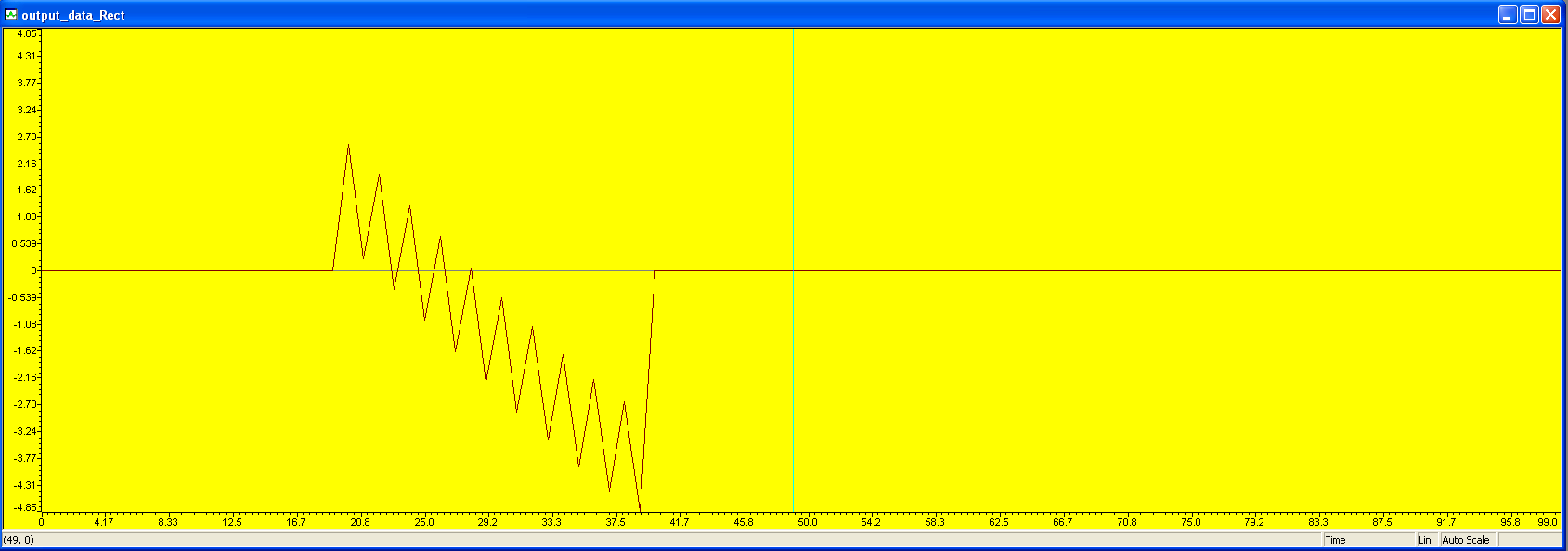


Рисунок 2 – Выходные данные (временная область)

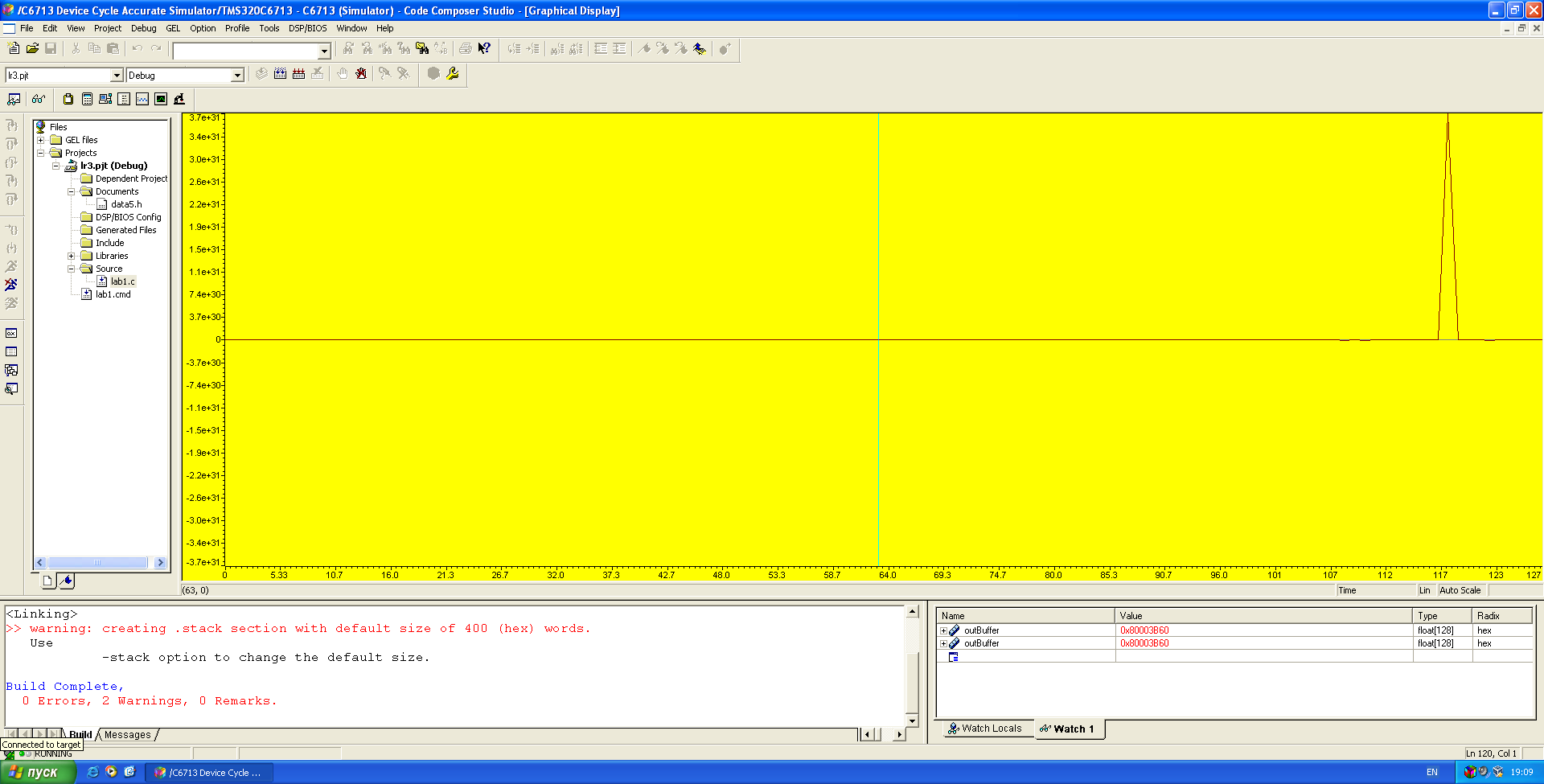


Рисунок 3 – Спектр входного сигнала (в разах)

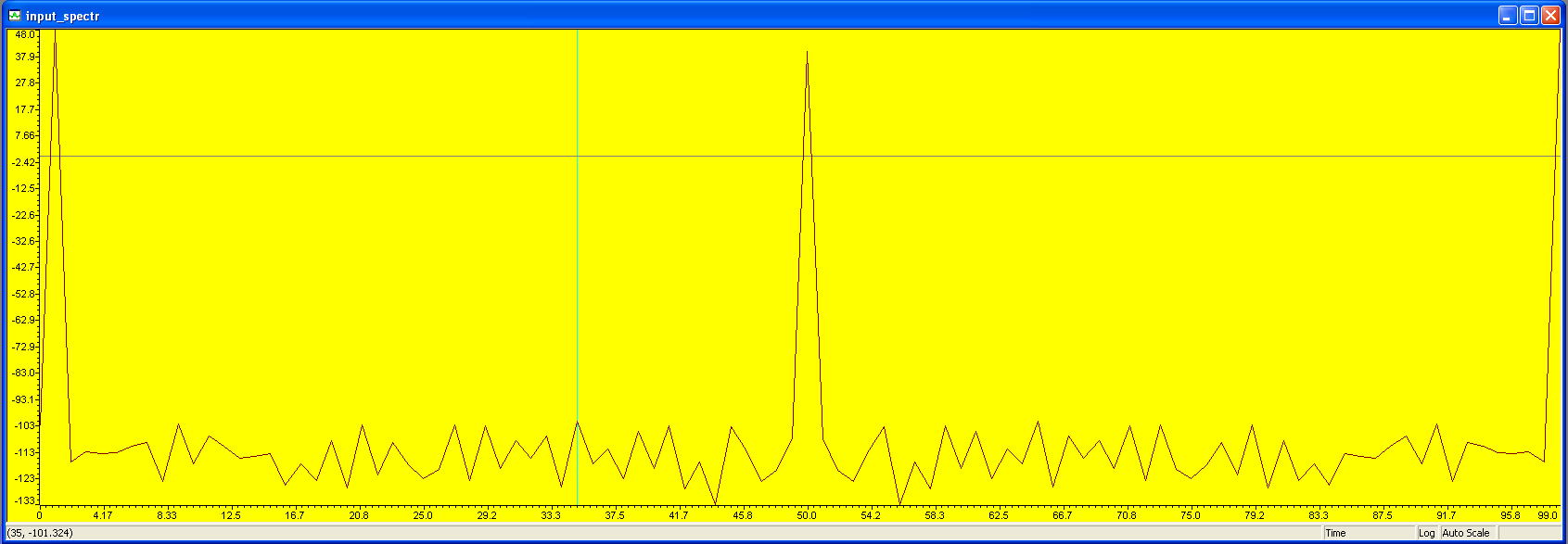


Рисунок 4 – Спектр входного сигнала (в дБ)

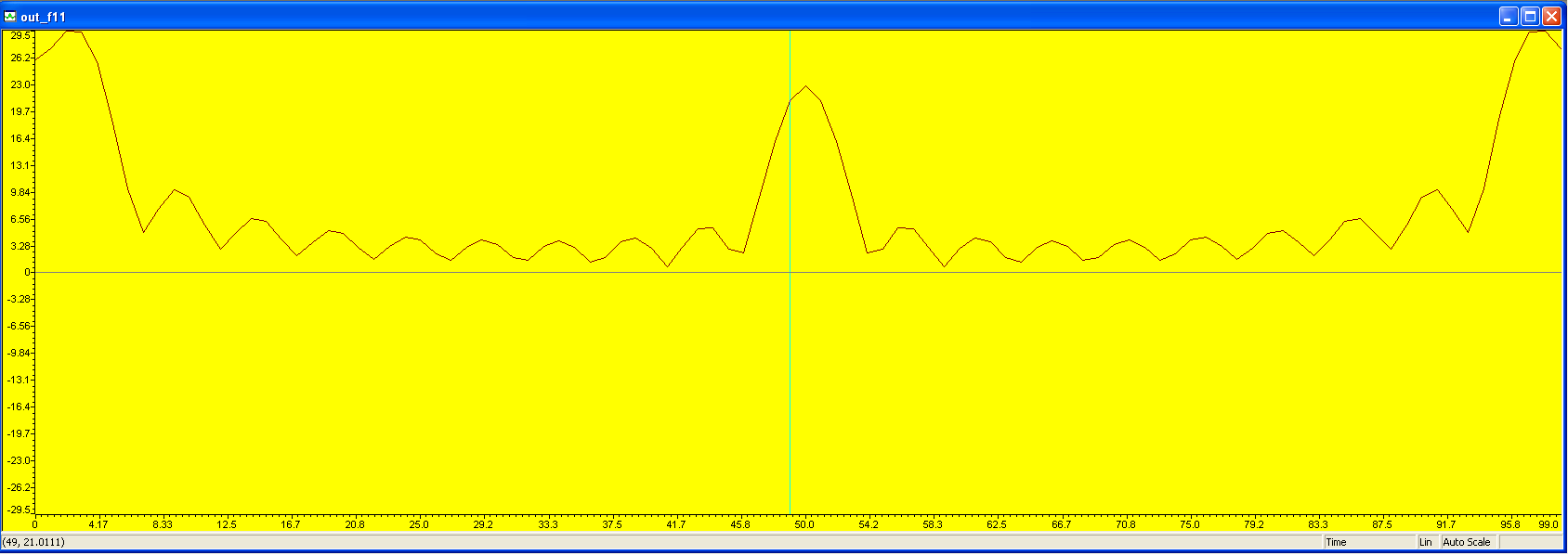


Рисунок 5 – Спектр выходного сигнала (в разах)

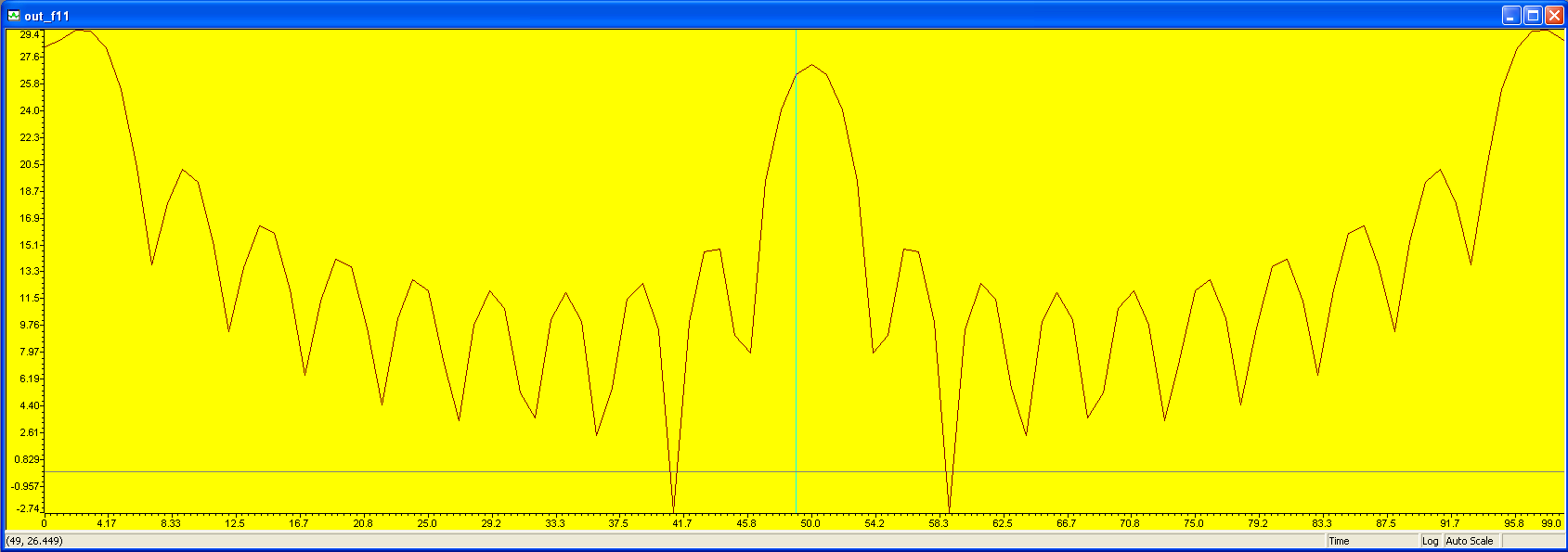


Рисунок 6 – Спектр выходного сигнала (в дБ)

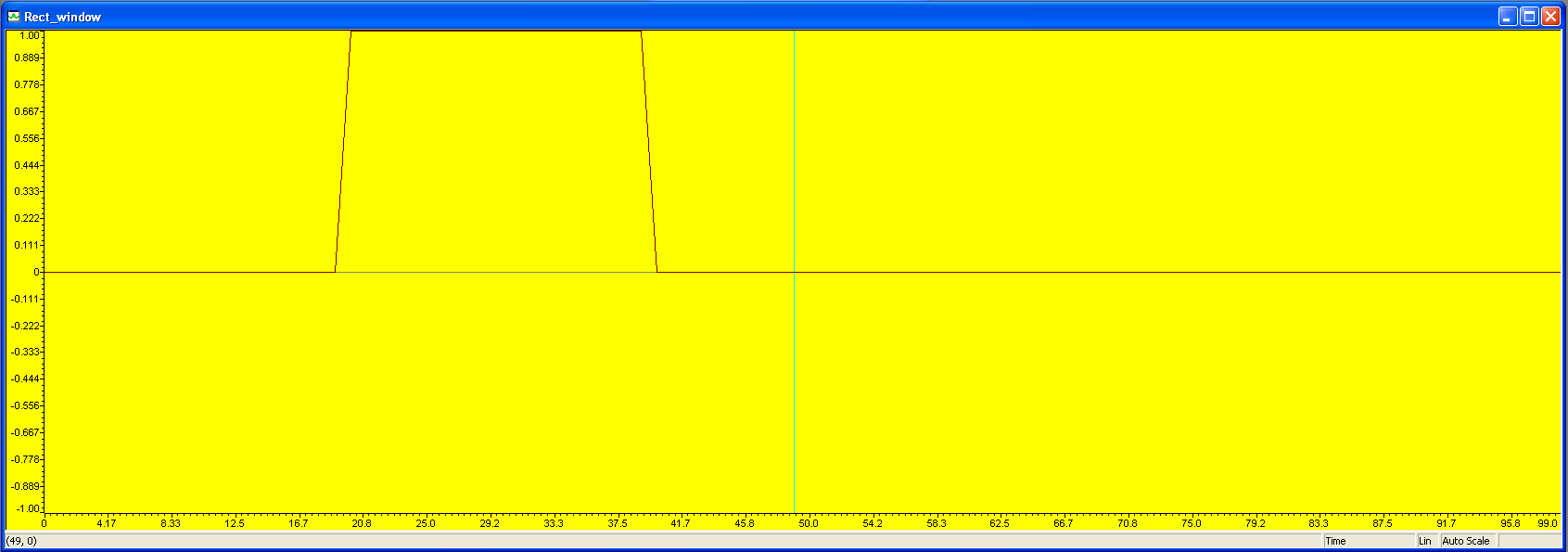


Рисунок 7 – График прямоугольного окна (временная область)

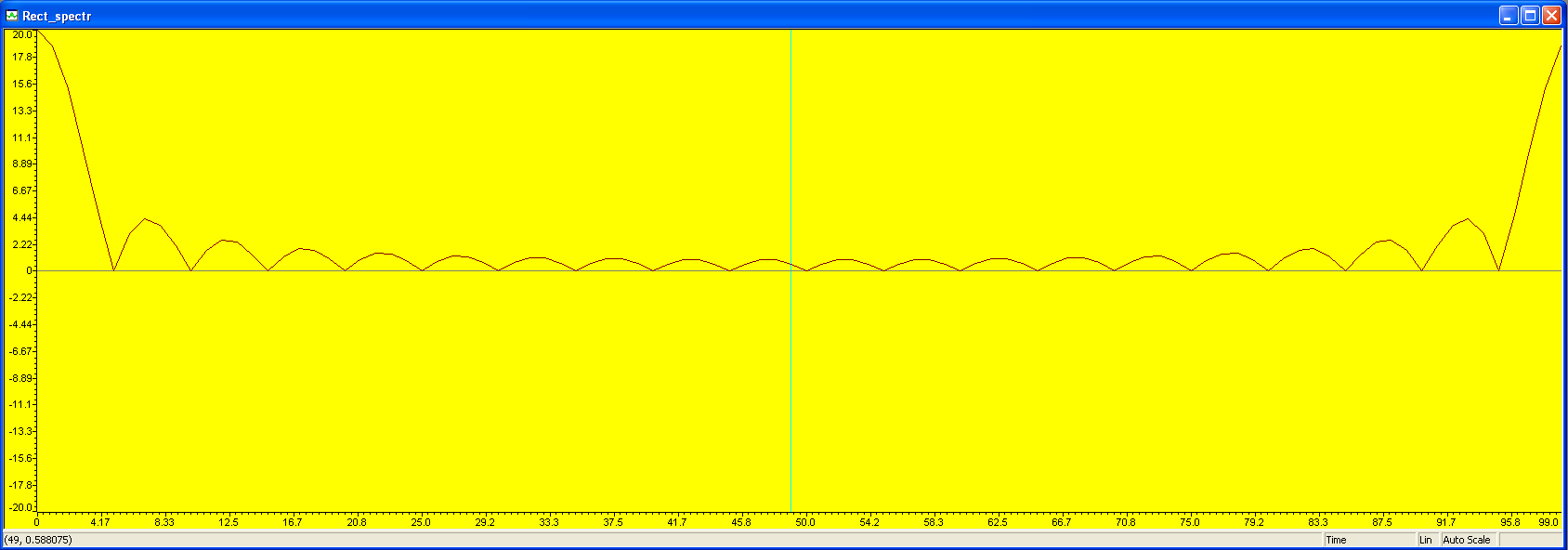




Рисунок 8 – Спектр прямоугольного окна (в разах) (желтый – результат вычисления программы, зеленый – авторассчет Code Composer Studio)

Вывод: графики спектра прямоугольного окна, рассчитанные написанной программой и внутренней функцией Code Composer Studio, сходятся. Желтый график отзеркаливается относительно центра, потому что рассчитывался на интервале 0…fs (fs - частота дискретизации), а зеленый – 0…0.5fs.

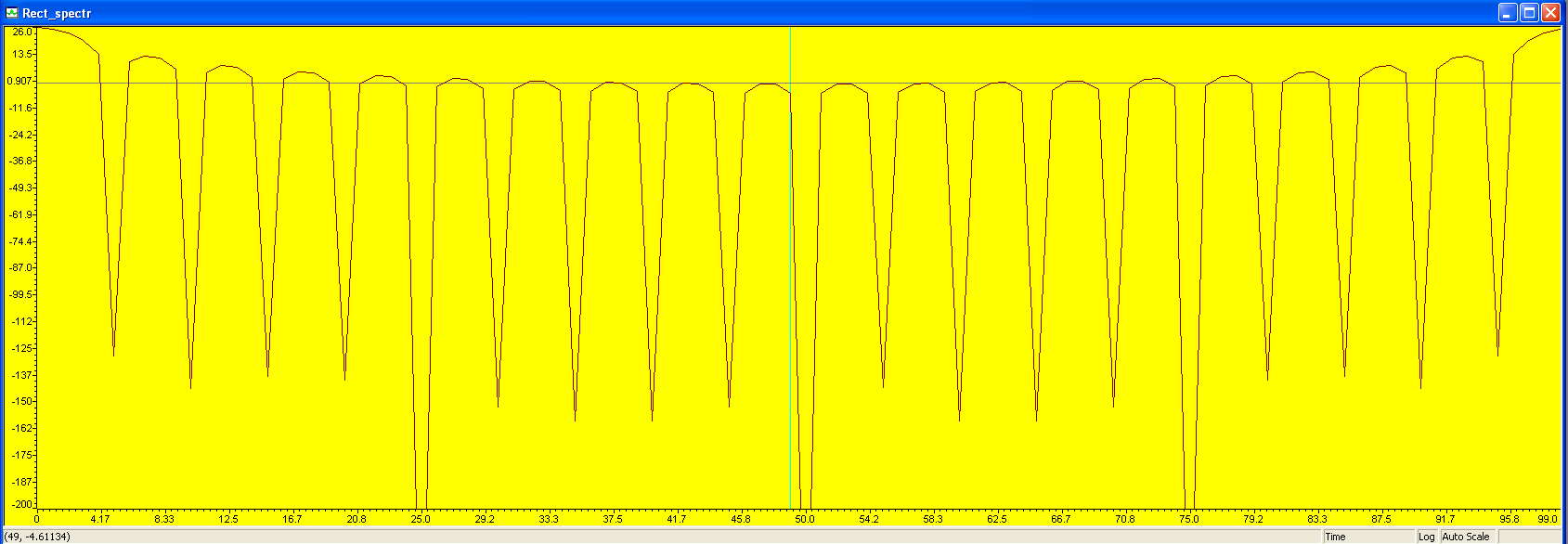


Рисунок 9 – Спектр прямоугольного окна (в дБ)

2) **Изучение Гауссовского окна**

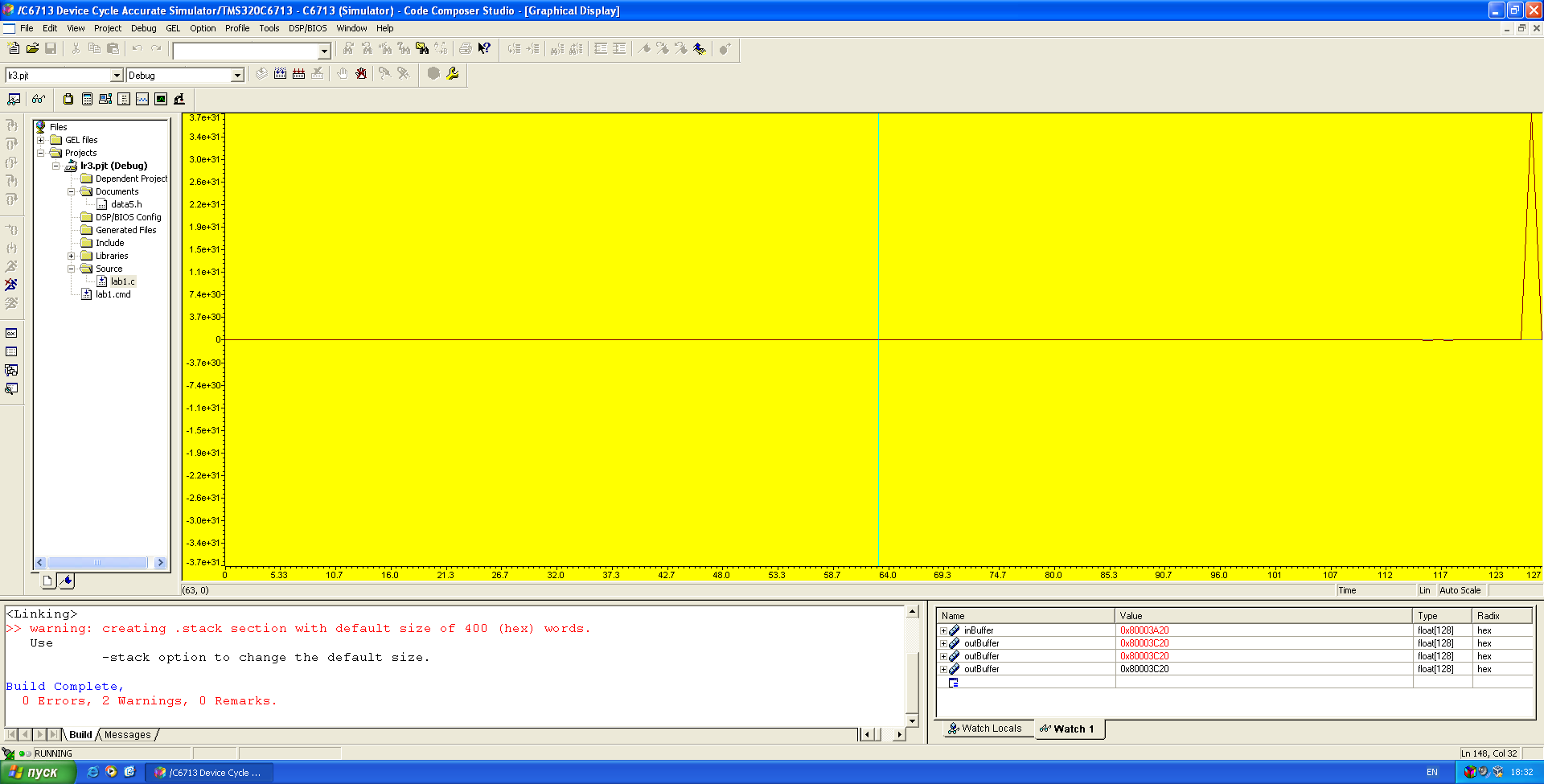


Рисунок 10 – Спектр входного сигнала

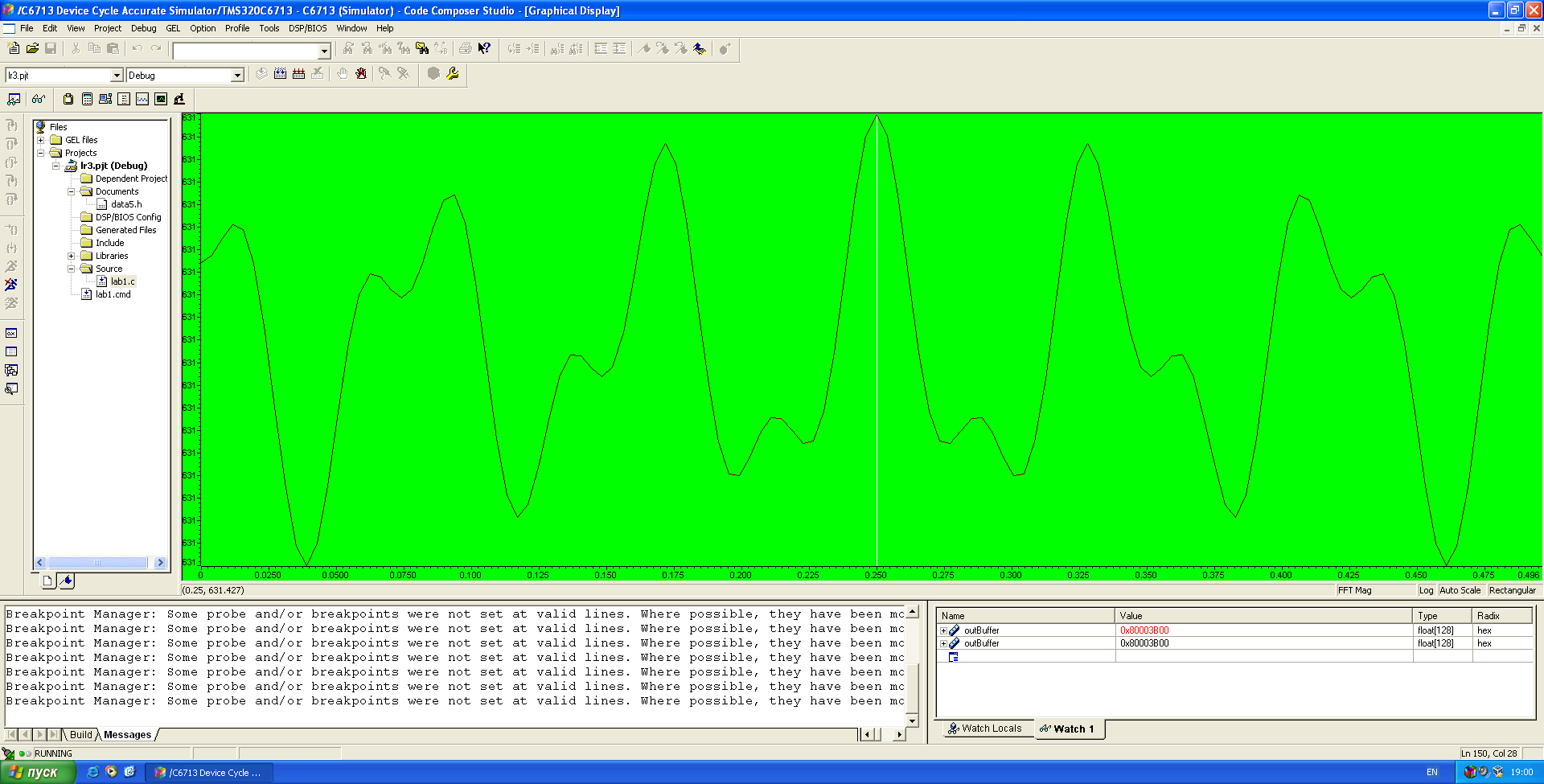


Рисунок 11 – Спектр выходного сигнала

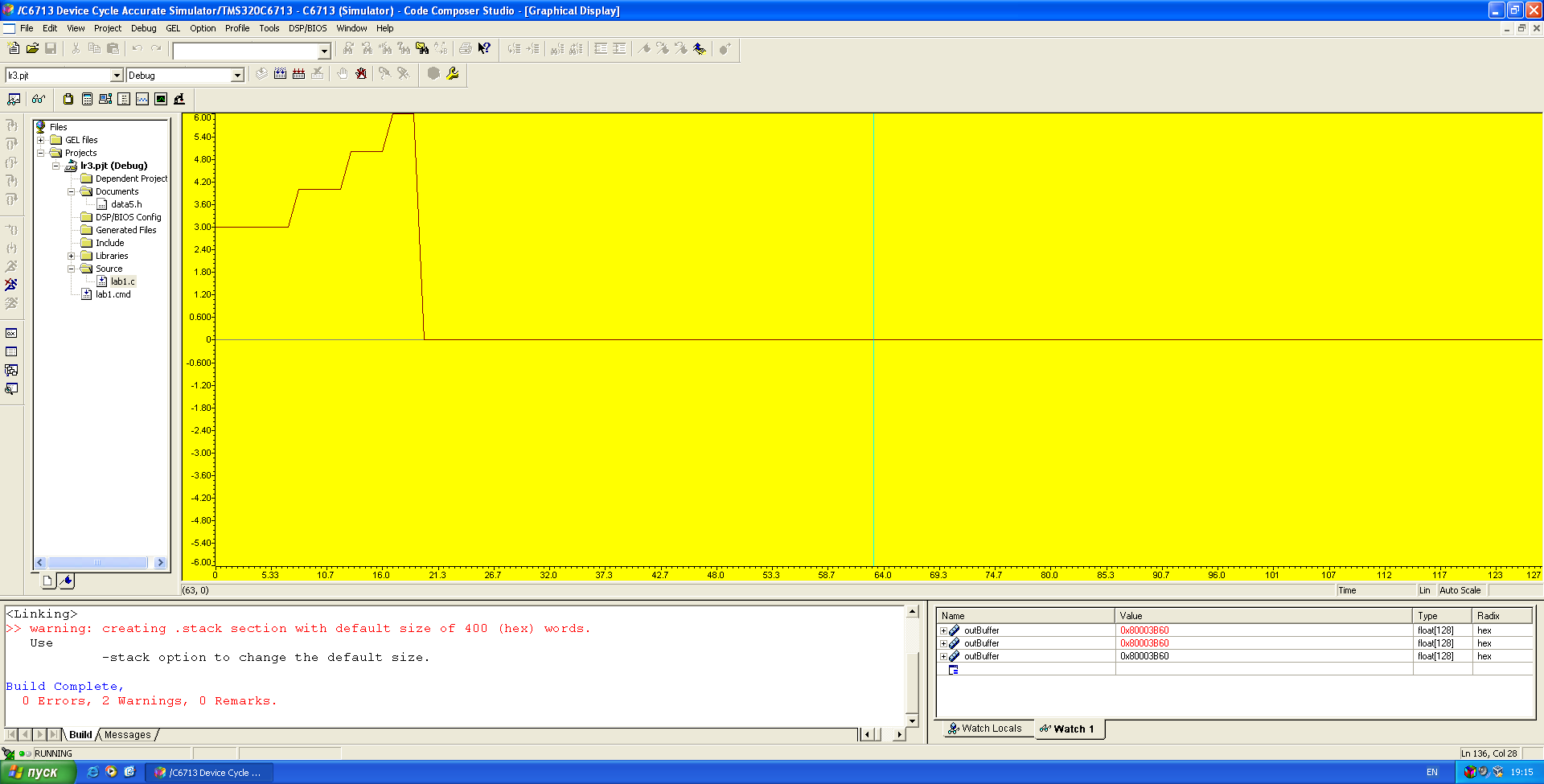


Рисунок 12 – График Гауссовского окна

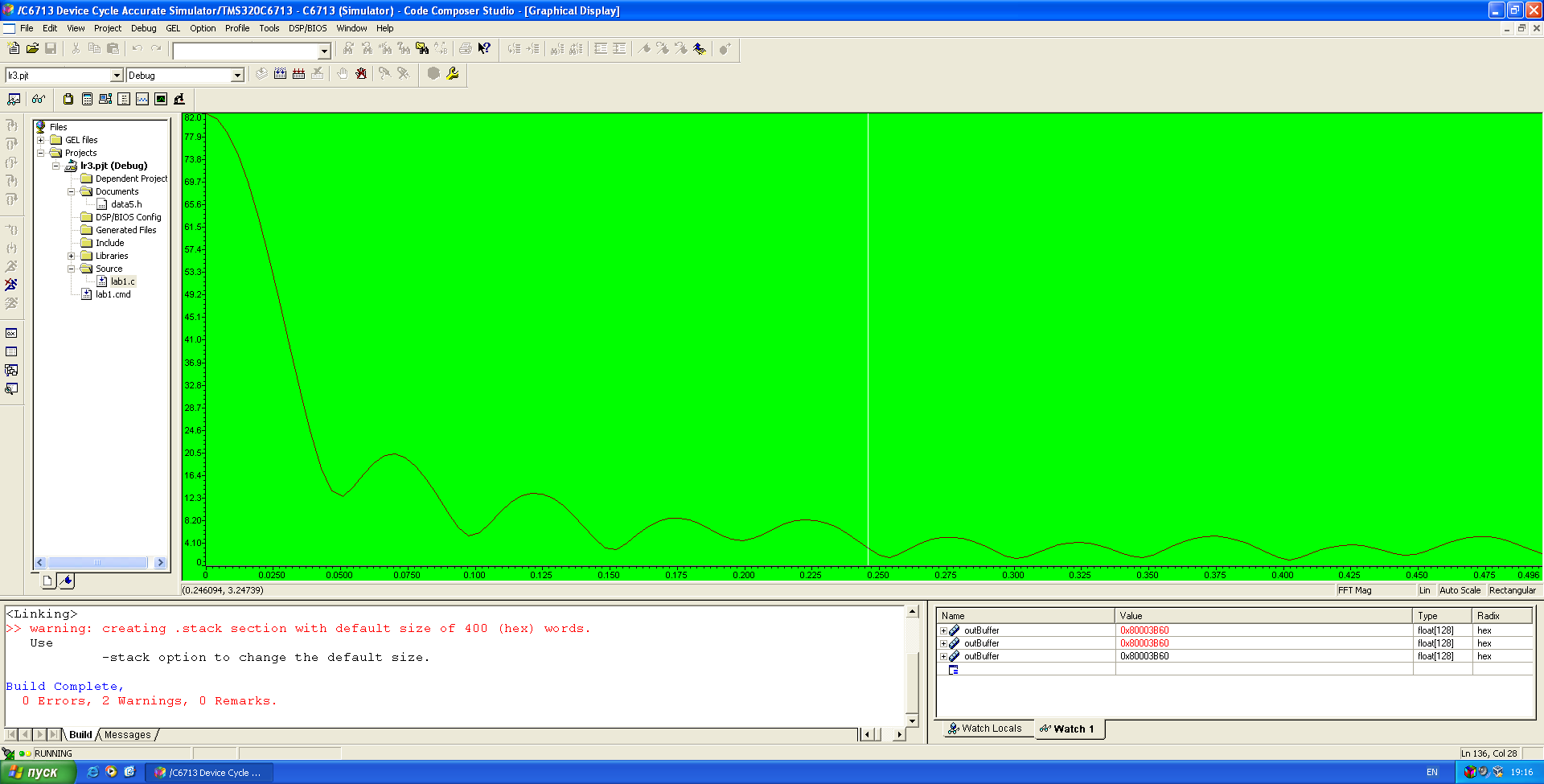


Рисунок 13 – Спектр выходного сигнала

3) **Изучение окна Бартлетта**

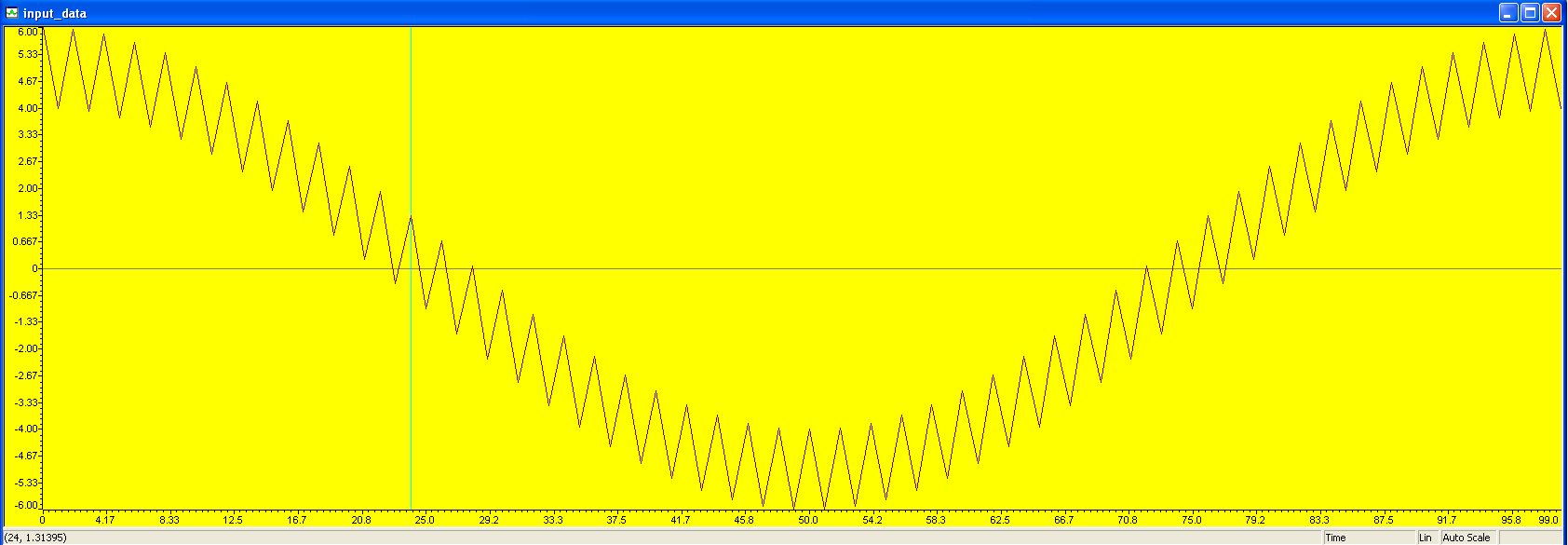


Рисунок 14 – Входные данные (временная область)

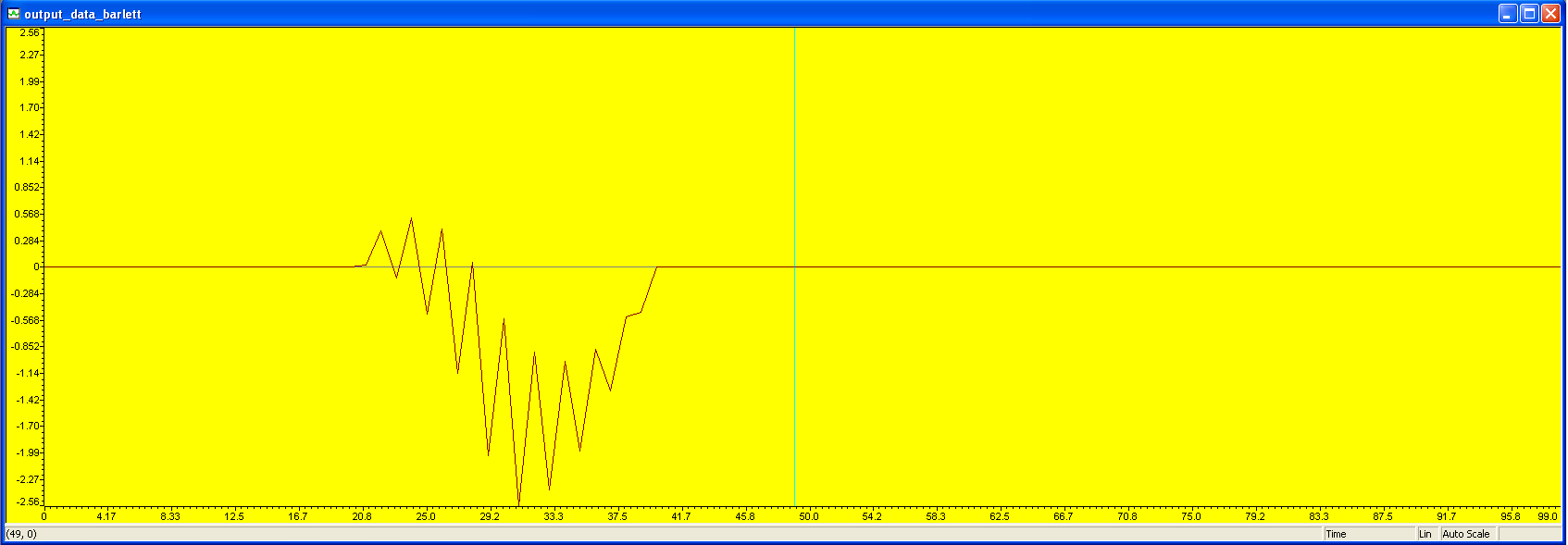


Рисунок 15 – Выходные данные (временная область)

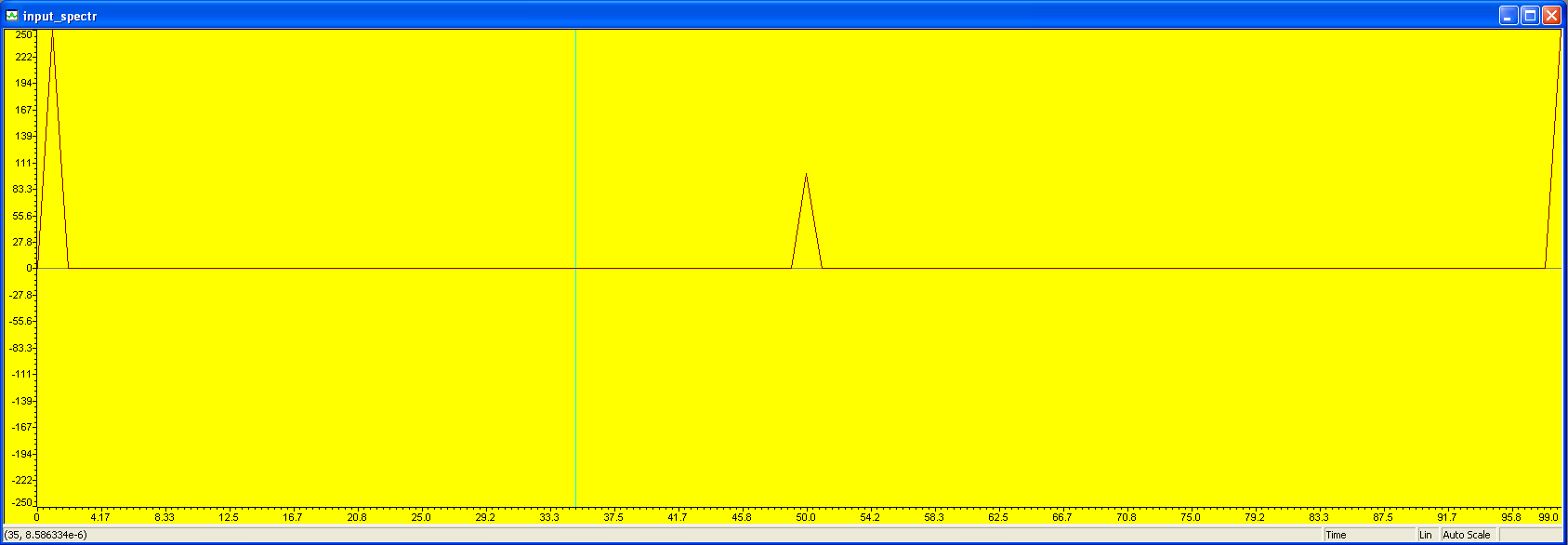


Рисунок 16 – Спектр входного сигнала (в разах)

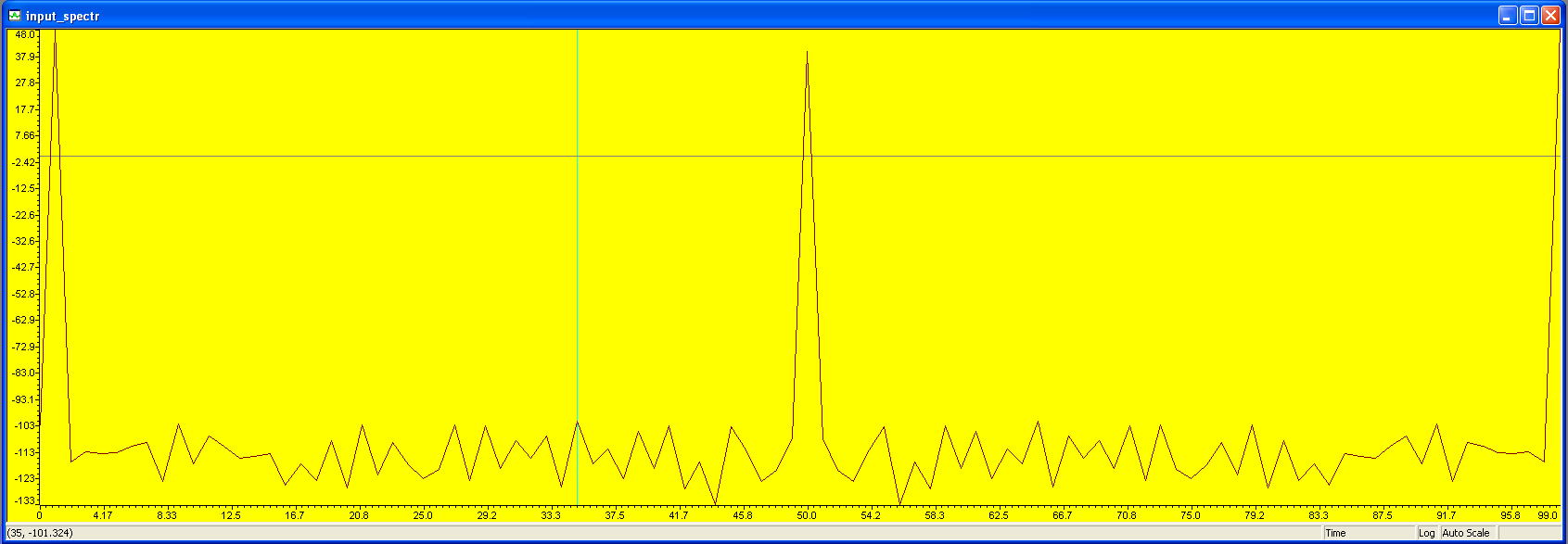


Рисунок 17 – Спектр входного сигнала (в дБ)

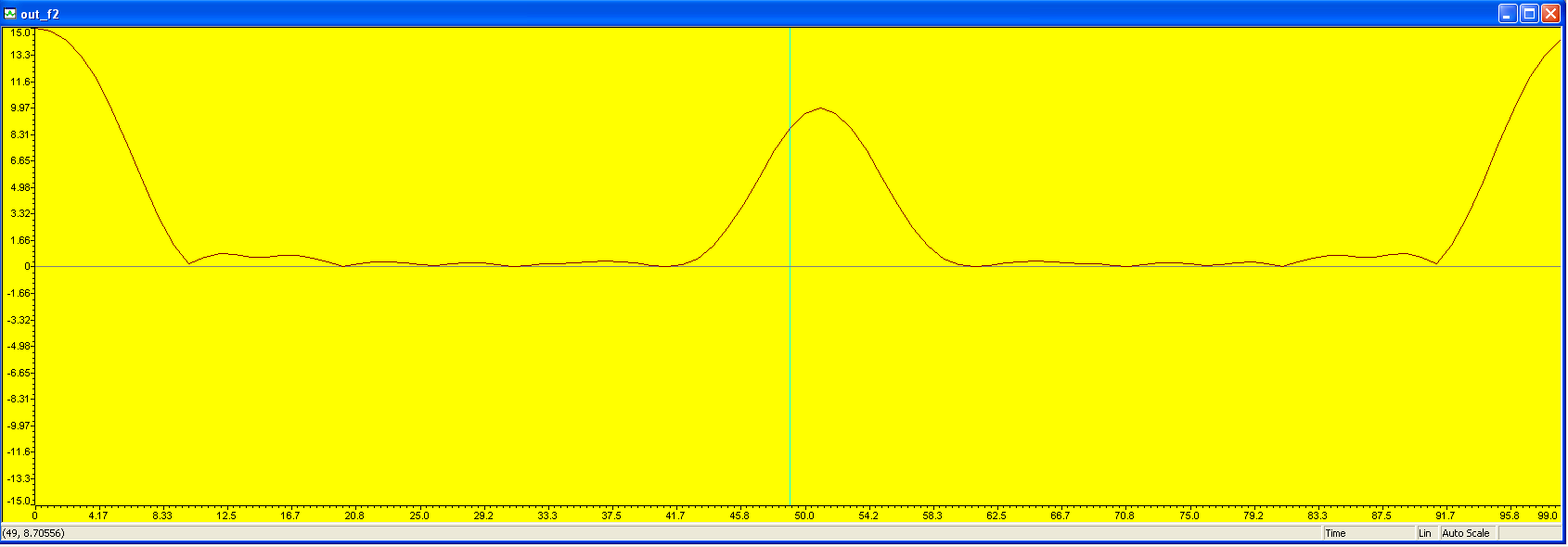


Рисунок 18 – Спектр выходного сигнала (в разах)

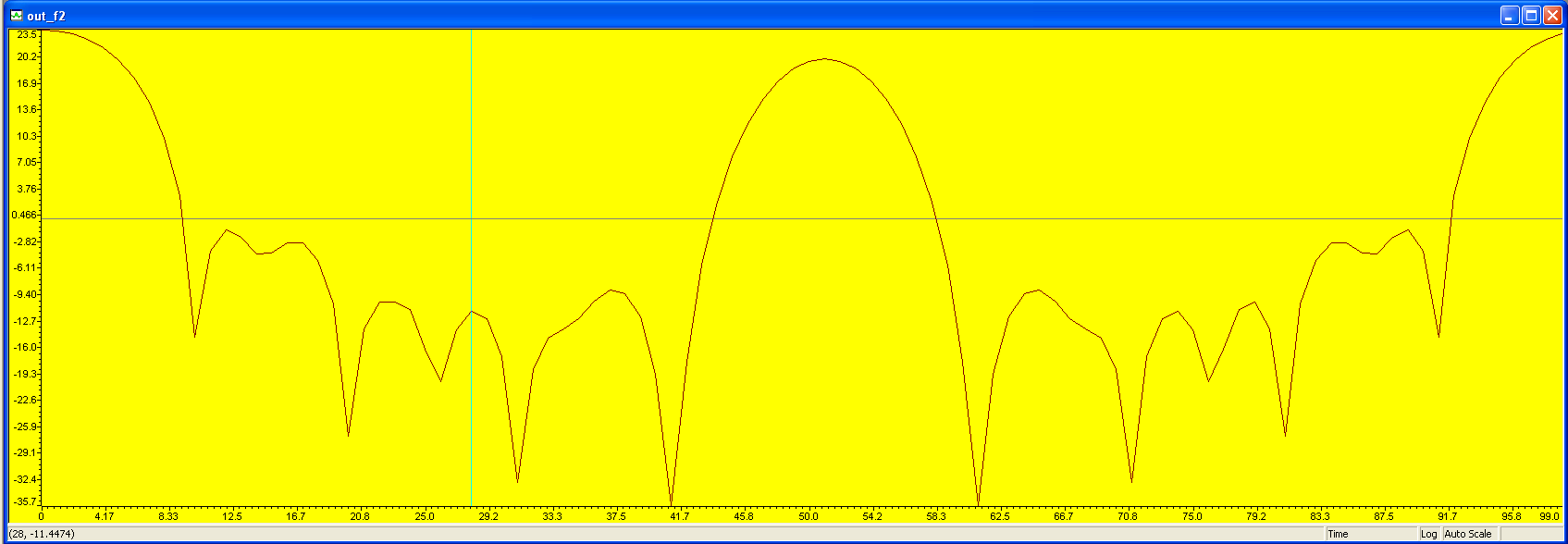


Рисунок 19 – Спектр выходного сигнала (в дБ)

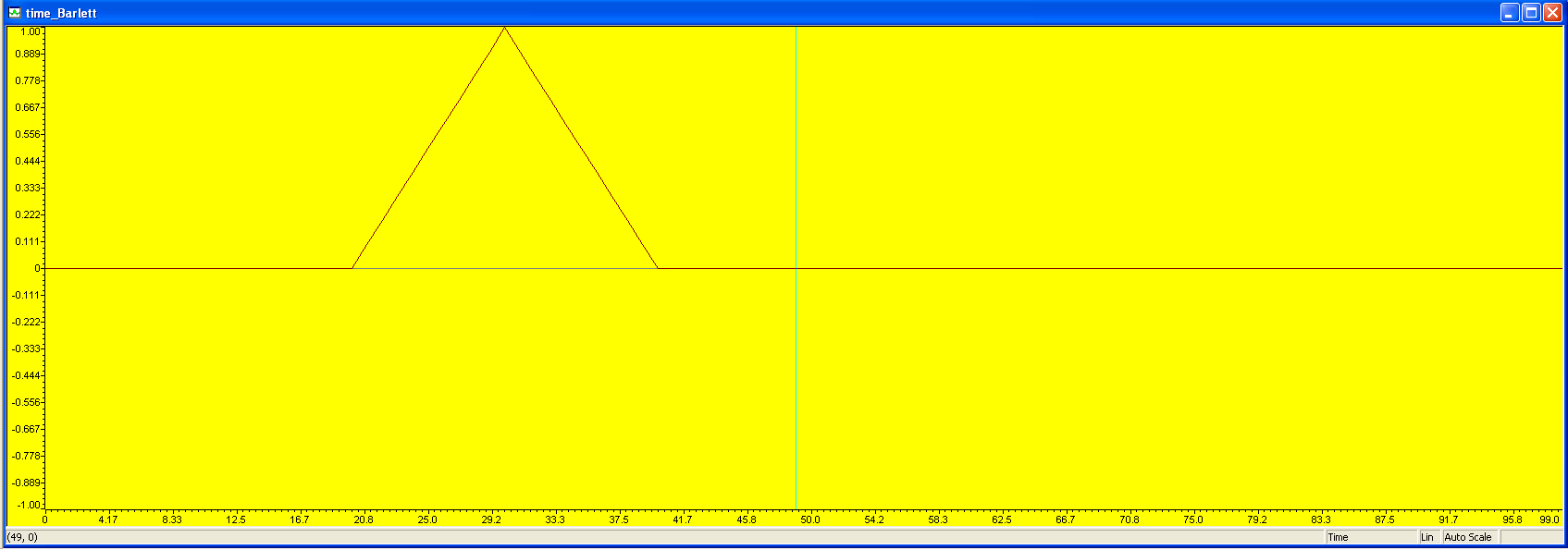
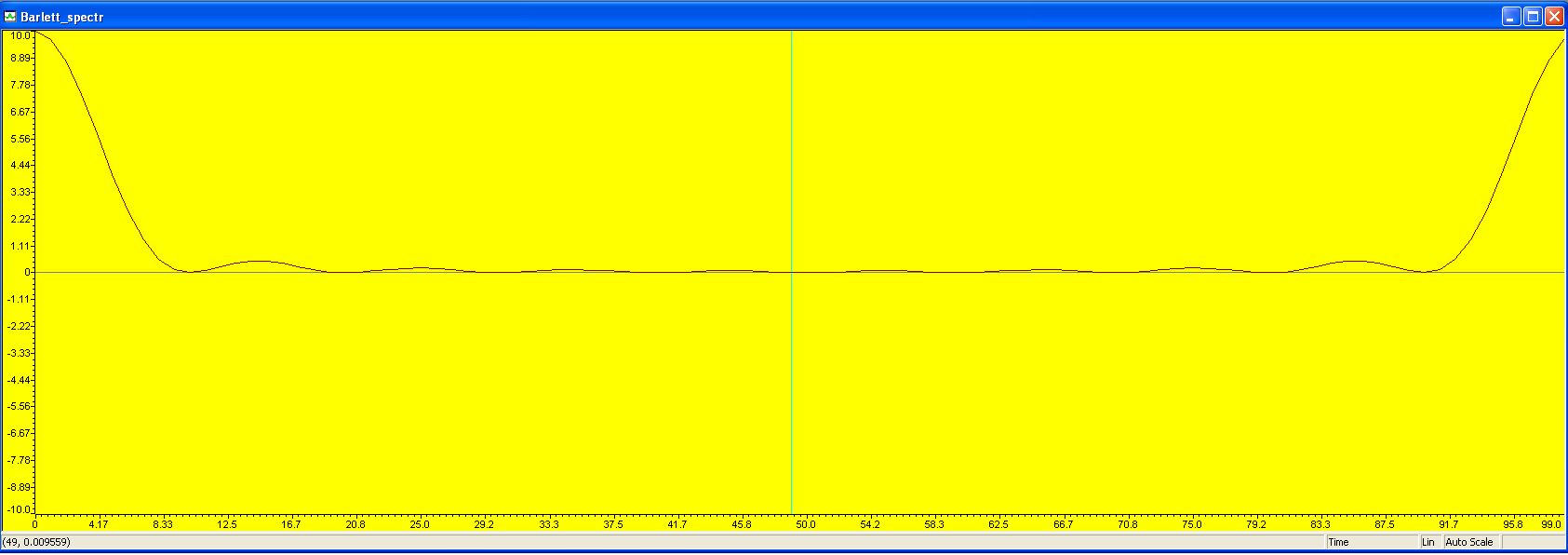


Рисунок 20 – График окна Бартлетта (временная область)



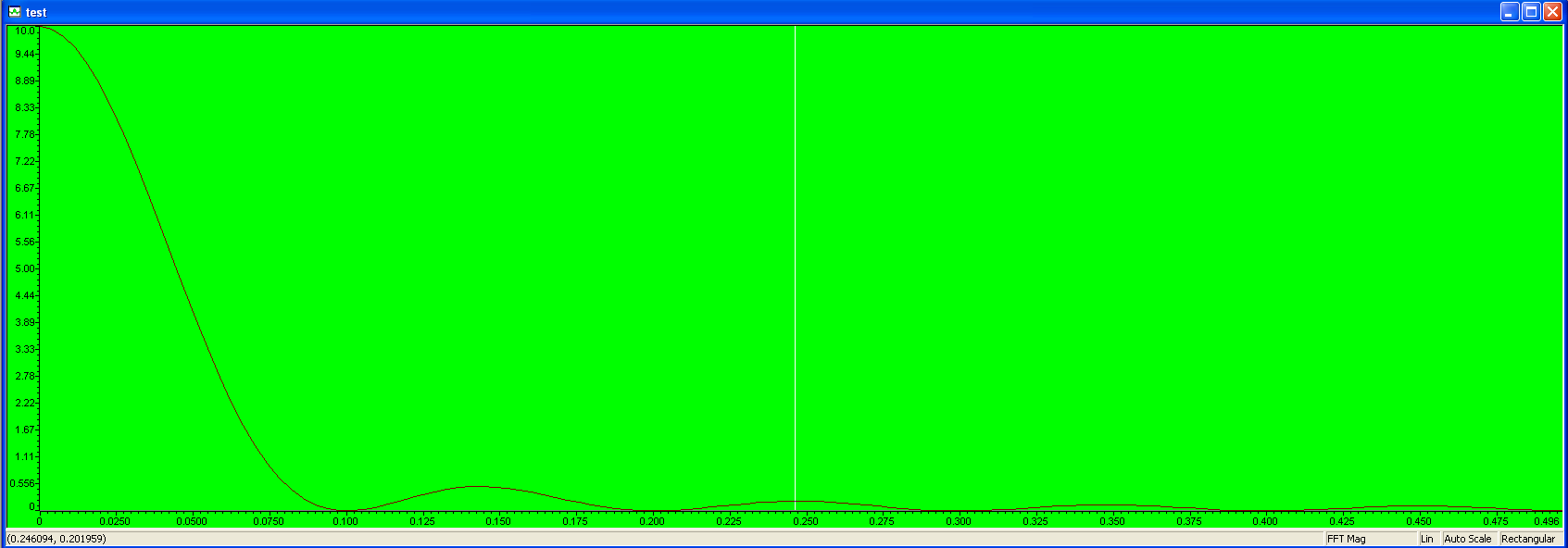


Рисунок 21 – Спектр окна Бартлетта (в разах) (желтый – результат вычисления программы, зеленый – авторассчет Code Composer Studio)

Вывод: графики спектра окна Бартлетта, рассчитанные написанной программой и внутренней функцией Code Composer Studio, сходятся. Желтый график отзеркаливается относительно центра, потому что рассчитывался на интервале 0…fs (fs - частота дискретизации), а зеленый – 0…0.5fs.

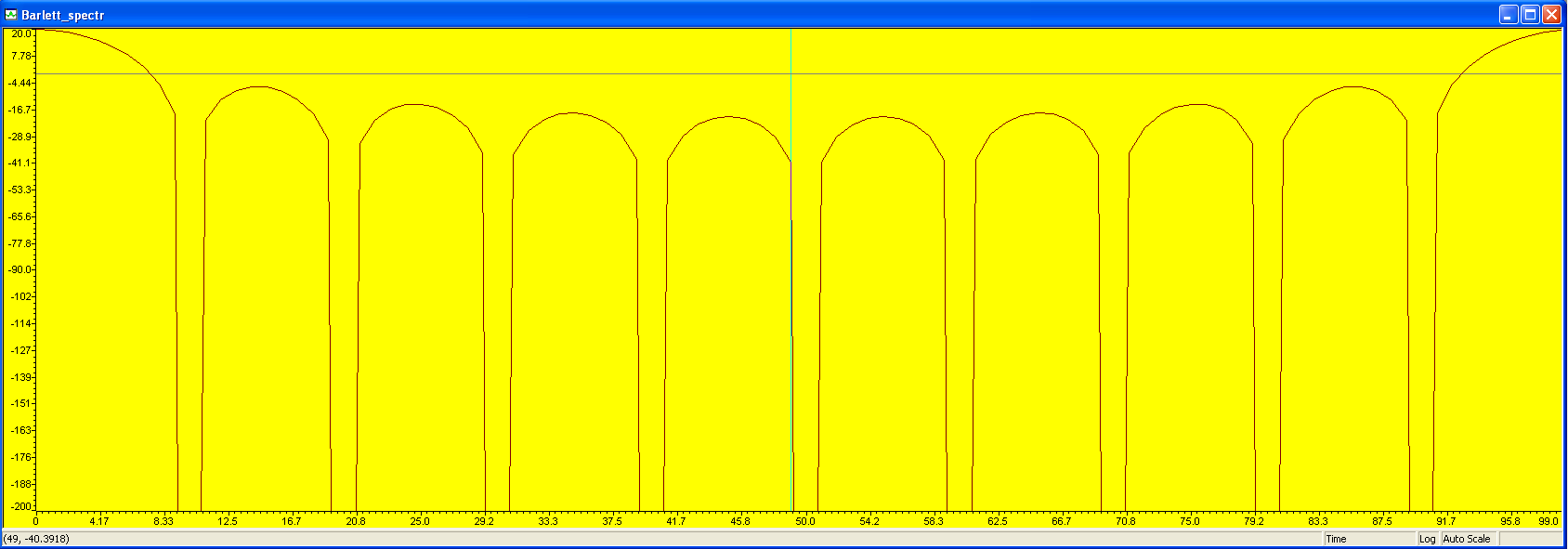


Рисунок 22 – Спектр окна Бартлетта (в дБ)

Выводы: острые спектральные пики ДПФ синусоиды расширились за счет воздействия копий преобразования окна. Минимальная ширина спектральных пиков взвешенной окном последовательности равна ширине главного лепестка. Боковые лепестки преобразования окна будут изменять амплитуды соседних спектральных пиков и могут маскировать присутствие слабых сигналов.

Текст программы:

#include "data5.h"

#include <stdio.h>

#include <stdlib.h>

#include <math.h>

#define MAX\_SIZE 128

#define PI 3.141592653589793

#define N 20

float inBuffer[MAX\_SIZE] = {A\_ARRAY};

float outBuffer[MAX\_SIZE];

struct complex {

float real;

float image;

};

struct complex dft(float\* x, int n, int k) {

int i = 0;

float resReal = 0;

float resImage = 0;

struct complex result;

result.real = 0;

result.image = 0;

for(i = 0; i < n; i++) {

resReal += (x[i] \* cos(2 \* PI \* i \* k / n));

resImage -= (x[i] \* sin(2 \* PI \* i \* k / n));

}

result.real = resReal;

result.image = resImage;

return result;

}

//dop functionality

struct complex complexMul(struct complex a, struct complex b) {

struct complex result;

result.real = a.real \* b.real - a.image \* b.image;

result.image = a.real \* b.image + a.image \* b.real;

return result;

}

//dop functionality

struct complex complexAdd(struct complex a, struct complex b) {

struct complex result;

result.real = a.real + b.real;

result.image = a.image + b.image;

return result;

}

//dop functionality

float idft(struct complex\* x, int n, int k) {

float result = 0;

struct complex temp;

struct complex resTemp;

int i = 0;

resTemp.real = 0;

resTemp.image = 0;

for(i = 0; i < n; i++) {

temp.real = cos(2 \* PI \* i \* k / n);

temp.image = sin(2 \* PI \* i \* k /n);

temp = complexMul(x[i], temp);

resTemp = complexAdd(resTemp, temp);

}

result = resTemp.real + resTemp.image;

result /= n;

return result;

}

//achx of spectr

float curve(struct complex x) {

return sqrt(x.real \* x.real + x.image \* x.image);

}

//rectangle window

void rect(float\* inputData, float\* outputData, int n) {

int i = 0;

for(i = 0; i < n; i++) {

if(i < N) {

outputData[i] = 1;

} else {

outputData[i] = 0;

}

outputData[i] \*= inputData[i];

}

}

//gaussing window

void gaussing(float\* inputData, float\* outputData, int n) {

int i = 0;

for(i = 0; i < n; i++) {

if(i < N) {

outputData[i] = exp(-12.5 \* pow((i - (N - 1) / 2) / (N - 1), 2));

} else {

outputData[i] = 0;

}

outputData[i] \*= inputData[i];

}

}

//barlett window

void barlett(float\* inputData, float\* outputData, int n) {

int i = 0;

for(i = 0; i < n; i++) {

outputData[i] = 0;

if(i < N) {

outputData[i] = 1 - 2 \* abs(i - (N - 1) / 2) / (N - 1);

}

outputData[i] \*= inputData[i];

}

}

void main() {

int k = 0;

struct complex spectrSignal;

float arrToShow[MAX\_SIZE];

//spectr of input signal

for(k = 0; k < MAX\_SIZE; k++) {

spectrSignal = dft(inBuffer, MAX\_SIZE, k);

arrToShow[k] = curve(spectrSignal);

}

//for normal view

arrToShow[MAX\_SIZE - 1] += 1;

arrToShow[MAX\_SIZE - 1] -= 1;

k++;//for breakpoint

//rectangle window

rect(inBuffer, outBuffer, MAX\_SIZE);

spectr of signal with window

for(k = 0; k < MAX\_SIZE; k++) {

spectrSignal = dft(outBuffer, MAX\_SIZE, k);

arrToShow[k] = curve(spectrSignal);

}

for normal view

arrToShow[MAX\_SIZE - 1] += 1;

arrToShow[MAX\_SIZE - 1] -= 1;

k++;//for breakpoint

//gaussian window

gaussing(inBuffer, outBuffer, MAX\_SIZE);

//spectr of signal with window

for(k = 0; k < MAX\_SIZE; k++) {

spectrSignal = dft(outBuffer, MAX\_SIZE, k);

arrToShow[k] = curve(spectrSignal);

}

for normal view

arrToShow[MAX\_SIZE - 1] += 1;

arrToShow[MAX\_SIZE - 1] -= 1;

k++;//for breakpoint

//barlett window

barlett(inBuffer, outBuffer, MAX\_SIZE);

//spectr of signal with window

for(k = 0; k < MAX\_SIZE; k++) {

spectrSignal = dft(outBuffer, MAX\_SIZE, k);

arrToShow[k] = curve(spectrSignal);

}

//for normal view

arrToShow[MAX\_SIZE - 1] += 1;

arrToShow[MAX\_SIZE - 1] -= 1;

k++;//for breakpoint