Digital Signal Processing

Lab 2 report

Marco Casagrande – Julius Rauscher | IE6-DSP | 25.4.17

Contents

[Attachment A 2](#_Toc481848645)

[Calculations 2](#_Toc481848646)

[Attachment B 3](#_Toc481848647)

[Input sequence 3](#_Toc481848648)

[Calulation results 3](#_Toc481848649)

[MATLAB results 4](#_Toc481848650)

[MATLAB script 4](#_Toc481848651)

[Comments 5](#_Toc481848652)

[Attachment C 5](#_Toc481848653)

[Input sequence 5](#_Toc481848654)

[Calulation results 5](#_Toc481848655)

[MATLAB results 5](#_Toc481848656)

[MATLAB script 6](#_Toc481848657)

[Comments 7](#_Toc481848658)

[Question a) 7](#_Toc481848659)

[Question b) 7](#_Toc481848660)

[Question c) 7](#_Toc481848661)

[Attachment D 8](#_Toc481848662)

[Input sequence 8](#_Toc481848663)

[DSK6713 output - overflow 8](#_Toc481848664)

[Butterfly rescaling implementation 8](#_Toc481848665)

[DSK6713 output – no overflow 9](#_Toc481848666)

[Attachment E 10](#_Toc481848667)

[Attachment F 12](#_Toc481848668)

[Attachment G 19](#_Toc481848669)

# Attachment A

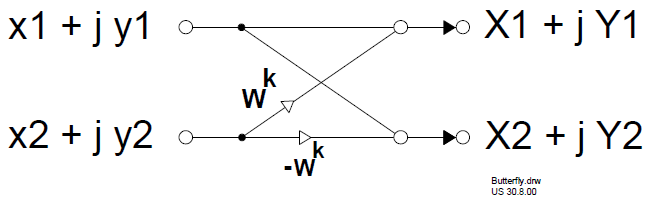


Figure - Butterfly structure

## Calculations

(x1 + j y1) + Wk (x2 + jy2) = X1 + j Y1

X1 = x1 + x2 cos(2 π k / N) + y2 sin(2 π k / N)

Y1 = y1 + y2 cos(2 π k / N) - x2 sin(2 π k / N)

(x2 + j y2) (-Wk) + (x1 + j y1) = X2 + j Y2

X2 = x1 – x2 cos(2 π k / N) – y2 sin(2 π k / N)

Y2 = y1 – y2 cos (2 π k / N) + x2 sin(2 π k / N)

X2 = -X1 + 2 x1

Y2 = -Y1 + 2 y1

# Attachment B

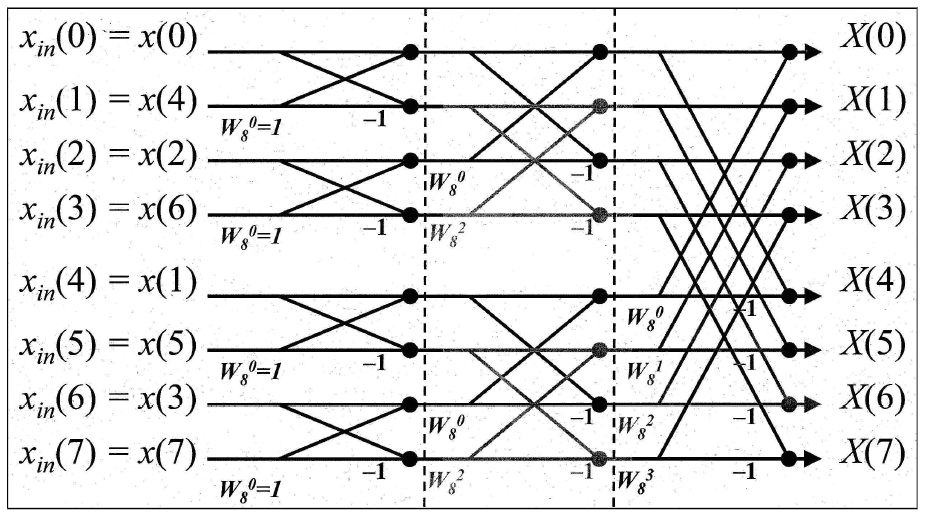


Figure - 8 points FFT structure

## Input sequence

x[n] = { 2000 0 -2000 0 2000 0 -2000 0}

## Calulation results

|  |  |
| --- | --- |
| INPUT STAGE  xin(0) = x(0) = 2000  xin(1) = x(4) = 2000  xin(2) = x(2) = -2000  xin(3) = x(6) = -2000  xin(4) = x(1) = 0  xin(5) = x(5) = 0  xin(6) = x(3) = 0  xin(7) = x(7) = 0 | 2ND STAGE  X2nd(0) = X1st(0) + W80 X1st(2) = 0  X2nd(1) = X1st(1) + W82 X1st(3) = 0  X2nd(2) = X1st(0) - W80 X1st(2) = 8000  X2nd(3) = X1st(1) - W82 X1st(3) = 0  X2nd(4) = X1st(4) + W80 X1st(6) = 0  X2nd(5) = X1st(5) + W82 X1st(7) = 0  X2nd(6) = X1st(4) - W80 X1st(6) = 0  X2nd(7) = X1st(5) - W82 X1st(7) = 0 |
| 1ST STAGE  X1st(0) = xin(0) + W80 xin(1) = 4000  X1st(1) = xin(0) - W80 xin(1) = 0  X1st(2) = xin(2) + W80 xin(3) = -4000  X1st(3) = xin(2) - W80 xin(3) = 0  X1st(4) = xin(4) + W80 xin(5) = 0  X1st(5) = xin(4) - W80 xin(5) = 0  X1st(6) = xin(6) + W80 xin(7) = 0  X1st(7) = xin(6) - W80 xin(7) = 0 | OUTPUT STAGE  X(0) = X2nd(0) + W80 X2nd(4) = 0  X(1) = X2nd(1) + W81 X2nd(5) = 0  X(2) = X2nd(2) + W82 X2nd(6) = 8000  X(3) = X2nd(3) + W83 X2nd(7) = 0  X(4) = X2nd(0) - W80 X2nd(4) = 0  X(5) = X2nd(1) - W81 X2nd(5) = 0  X(6) = X2nd(2) - W82 X2nd(6) = 8000  X(7) = X2nd(3) - W83 X2nd(7) = 0 |

## MATLAB results

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| k | Xin(k) | X1st(k) | X2nd(k) | X(k) |
| 0 | 2000 | 4000 | 0 | 0 |
| 1 | 2000 | 0 | 0 | 0 |
| 2 | -2000 | -4000 | 8000 | 8000 |
| 3 | -2000 | 0 | 0 | 0 |
| 4 | 0 | 0 | 0 | 0 |
| 5 | 0 | 0 | 0 | 0 |
| 6 | 0 | 0 | 0 | 8000 |
| 7 | 0 | 0 | 0 | 0 |

## MATLAB script

% FFT\_a.m

clear all;

N = 8;

n\_stages = log2(N);

x = [2000 0 -2000 0 2000 0 -2000 0];

X = [bitrevorder(x)' zeros(8,n\_stages)];

for stage = 1 : n\_stages

for group = 1 : N/(2^stage)

for butter = 1 : N/(2\*N/(2^stage))

s1 = (butter+(group-1)\*2^stage);

s2 = (butter+(group-1)\*2^stage)+2^(stage-1);

t\_exp = N/(2^(stage))\*(butter-1);

[X(s1, stage+1), X(s2, stage+1)] = butterfly(X(s1, stage), X(s2, stage), t\_exp, N);

end

end

end

function [out1, out2] = butterfly(in1, in2, twiddle\_exp, N)

out1 = in1 + in2 \* exp(-1i\*2\*pi/N\*twiddle\_exp);

out2 = in1 - in2 \* exp(-1i\*2\*pi/N\*twiddle\_exp);

end

## Comments

X(2) = X(6) = 8000 < Xshort, max = 215 – 1 = 32767

No overflow occurs

# Attachment C

## Input sequence

x[n] = { 10000 0 -10000 0 10000 0 -10000 0}

## Calulation results

|  |  |
| --- | --- |
| INPUT STAGE  xin(0) = x(0) = 10000  xin(1) = x(4) = 10000  xin(2) = x(2) = -10000  xin(3) = x(6) = -10000  xin(4) = x(1) = 0  xin(5) = x(5) = 0  xin(6) = x(3) = 0  xin(7) = x(7) = 0 | 2ND STAGE  X2nd(0) = X1st(0) + W80 X1st(2) = 0  X2nd(1) = X1st(1) + W82 X1st(3) = 0  X2nd(2) = X1st(0) - W80 X1st(2) = 40000  X2nd(3) = X1st(1) - W82 X1st(3) = 0  X2nd(4) = X1st(4) + W80 X1st(6) = 0  X2nd(5) = X1st(5) + W82 X1st(7) = 0  X2nd(6) = X1st(4) - W80 X1st(6) = 0  X2nd(7) = X1st(5) - W82 X1st(7) = 0 |
| 1ST STAGE  X1st(0) = xin(0) + W80 xin(1) = 20000  X1st(1) = xin(0) - W80 xin(1) = 0  X1st(2) = xin(2) + W80 xin(3) = -20000  X1st(3) = xin(2) - W80 xin(3) = 0  X1st(4) = xin(4) + W80 xin(5) = 0  X1st(5) = xin(4) - W80 xin(5) = 0  X1st(6) = xin(6) + W80 xin(7) = 0  X1st(7) = xin(6) - W80 xin(7) = 0 | OUTPUT STAGE  X(0) = X2nd(0) + W80 X2nd(4) = 0  X(1) = X2nd(1) + W81 X2nd(5) = 0  X(2) = X2nd(2) + W82 X2nd(6) = 40000  X(3) = X2nd(3) + W83 X2nd(7) = 0  X(4) = X2nd(0) - W80 X2nd(4) = 0  X(5) = X2nd(1) - W81 X2nd(5) = 0  X(6) = X2nd(2) - W82 X2nd(6) = 40000  X(7) = X2nd(3) - W83 X2nd(7) = 0 |

## MATLAB results

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| k | Xin(k) | X1st(k) | X2nd(k) | X(k) |
| 0 | 10000 | 20000 | 0 | 0 |
| 1 | 10000 | 0 | 0 | 0 |
| 2 | -10000 | -20000 | 40000 | 40000 |
| 3 | -10000 | 0 | 0 | 0 |
| 4 | 0 | 0 | 0 | 0 |
| 5 | 0 | 0 | 0 | 0 |
| 6 | 0 | 0 | 0 | 40000 |
| 7 | 0 | 0 | 0 | 0 |

## MATLAB script

% FFT\_b.m

clear all;

N = 8;

n\_stages = log2(N);

x = [10000 0 -10000 0 10000 0 -10000 0];

X = [bitrevorder(x)' zeros(8,n\_stages)];

for stage = 1 : n\_stages

for group = 1 : N/(2^stage)

for butter = 1 : N/(2\*N/(2^stage))

s1 = (butter+(group-1)\*2^stage);

s2 = (butter+(group-1)\*2^stage)+2^(stage-1);

t\_exp = N/(2^(stage))\*(butter-1);

[X(s1, stage+1), X(s2, stage+1)] = butterfly(X(s1, stage), X(s2, stage), t\_exp, N);

end

end

end

function [out1, out2] = butterfly(in1, in2, twiddle\_exp, N)

out1 = in1 + in2 \* exp(-1i\*2\*pi/N\*twiddle\_exp);

out2 = in1 - in2 \* exp(-1i\*2\*pi/N\*twiddle\_exp);

end

## Comments

### Question a)

X2nd(2) = 40000 > Xshort, max = 215 – 1 = 32767

Overflow occurs!

(40000)10 = (0 1001 1100 0100 0000)2 => (1001 1100 0100 0000)2, short = (-25536)10

### Question b)

To avoid the overflow while computing the FFT with short integer 16 bits values without losing too much information in the process, we can shift the results of the intermediate sums at each stage by one position to the right, effectively halving it. By doing so, the result of the scaled sum is assured to be within the range of short int values, –32768 .. +32767.

**2-1**

**2-1**

X1st(k)

X1st(k+1)

X2nd(k+2)

**>>1**

**>>1**

1° STAGE

2° STAGE

Figure - Consecutive divisions each stage to avoid overflow

### Question c)

Using the approached described above, the FFT gain is compensated as the values at the output are effectively scaled by a factor of N = 2n\_stages , while the approximation is minimized in case odd values are halved. The even valued sums are rescaled without losing any information.

# Attachment D

## Input sequence

x[n] = { 10000 0 -10000 0 10000 0 -10000 0}

## DSK6713 output - overflow

0 0 - j 0

1 0 - j 0

2 -25539 - j 0

3 0 - j 0

4 0 - j 0

5 0 - j 0

6 -25536 - j 0

7 0 - j 0

As expected, the output matches the expectation listed in Attachment A and an overflow occurs.

## Butterfly rescaling implementation

The approach described in Attachment C is implemented in the provided file butterfly.c as follows:

//-----------------------------------------------------------

// Digital Signal Processing Lab

// FIR implementation in fixed-point arithmetic

//

// Filename : butterfly.c

//

// Author Svg 31. Aug 2005

//

// Description:

// This program computes the butterfly output signals IN PLACE

//

// \*p\_xa points to upper node of input, real part : x1

// \*p\_xb points to lower node of input, real part : x2

// \*(p\_xa+1) points to upper node of input, imag part : y1

// \*(p\_xb+1) points to lower node of input, imag part : y2

//

// Note : the cosine values are NEGATIVE !

// w1 = -real(wk), w2 = imag(wk)

//

// version 1 : checked 31.Aug. 05

void butterfly(short \*p\_xa, short \*p\_xb, short w1, short w2){

int xout\_r1\_temp;

int xout\_i1\_temp;

int xout\_r2\_temp;

int xout\_i2\_temp;

// upper node, real part

xout\_r1\_temp = ( \*p\_xa << 15 ) - \*p\_xb \* w1 - \*(p\_xb+1) \* w2 ;

// upper node, imaginary part

xout\_i1\_temp = ( \*(p\_xa+1)<< 15 ) + \*p\_xb \* w2 - \*(p\_xb+1) \* w1 ;

// lower node, real part

xout\_r2\_temp = ( \*p\_xa << 15 ) + \*p\_xb \* w1 + \*(p\_xb+1) \* w2 ;

// lower node, imaginary part

xout\_i2\_temp = ( \*(p\_xa+1)<< 15 ) - \*p\_xb \* w2 + \*(p\_xb+1) \* w1 ;

// Sum rescaling by shifting >>15 + 1 position the result of the sums

\*p\_xa = (short) (xout\_r1\_temp>>16); //write four results back IN-PLACE

\*(p\_xa+1) = (short) (xout\_i1\_temp>>16);

\*p\_xb = (short) (xout\_r2\_temp>>16);

\*(p\_xb+1) = (short) (xout\_i2\_temp>>16);

}

## DSK6713 output – no overflow

0 0 - j 0

1 0 - j 0

2 5000 - j 0

3 0 - j 0

4 0 - j 0

5 0 - j 0

6 5000 - j 0

7 0 - j 0

As expected, the results are rescaled with a factor of 1 / 8 and the overflow problem is solved.

# Attachment E

Essential for computing a 64-point FFT is the buffer size. This must be modified to 64 to store all the samples. With an increasing buffer size, the bit reversal must also generate 64 indexes compared to 8 before.

|  |  |
| --- | --- |
| 8-point FFT | 64-point FFT |
| #define N 8 | #define N 64 |
| // the bit-reversed values for N=8 !!  short index[] = {0, 4, 2, 6, 1, 5, 3, 7}; | // the bit-reversed values for N=64 !!  // ---------------------------------------------------  short index[] = {0, 32, 16, 48, 8, 40, 24, 56, 4, 36, 20, 52, 12, 44, 28, 60, 2, 34, 18, 50, 10, 42, 26, 58, 6, 38, 22, 54, 14, 46, 30, 62, 1, 33, 17, 49, 9, 41, 25, 57, 5, 37, 21, 53, 13, 45, 29, 61, 3, 35, 19, 51, 11, 43, 27, 59, 7, 39, 23, 55, 15, 47, 31, 63}; |

Additionally, the 64FFT\_Radix2 should also loop through 6 stages, as indicated below in bold:

void FFT64\_radix2(int N\_FFT, short x[], short w\_r[], short w\_i[]){

short k=0, t=0, j=0, stages;

short BF\_Span=1, BLK\_Step=2;

short \*p\_xa, \*p\_xb;

short num\_groups = N\_FFT/2, numBFY\_per\_group = 1;

short xb\_ofs, x\_step;

short y\_ofs;

**// Number of stages is ld(N) = log10(N)/log10(2).**

**// For N=64, stages = 6.**

**stages = 6;**

//---------- The (outer) k-loop is across THE STAGES ---------------

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

//---------- The middle) t-loop is across THE GROUPs ---------------

for (t = 0; t < num\_groups; t++) {

// p\_xa: ptr to UPPER butterfly input, real value

p\_xa = x;

// current span, \*2 because real and imag values are in ONE array x[2\*N]

xb\_ofs = 2\*BF\_Span;

// p\_xb: ptr to LOWER butterfly input, real value

p\_xb = x + xb\_ofs ;

// x\_step : the distance to the NEXT butterfly in THIS stage, multiplied

// by loop variable "t"

// again \*2 because real and imag values are in ONE array x[2\*N]

x\_step = t \* 2\*BLK\_Step;

// adapt p\_xa and p\_xb according to the number of butterflies per group

p\_xa += x\_step;

p\_xb += x\_step;

//---------- The (inner) j-loop is across THE BUTTERFLIES ----------

for (j=0; j < numBFY\_per\_group; j++){

y\_ofs = j\*num\_groups;

butterfly( p\_xa, p\_xb, w\_r[0 + y\_ofs], w\_i[0 + y\_ofs]);

// adapt index

p\_xa += 2;

p\_xb += 2;

} // j-loop ends

} // t-loop ends

// update parameters

num\_groups >>= 1;

numBFY\_per\_group <<= 1;

BLK\_Step <<= 1;

BF\_Span <<= 1;

} // k-loop ends

} // end of FFT64\_radix2-function

# Attachment F

We tested our 64-point algorithm with the following input sequences:

1. )

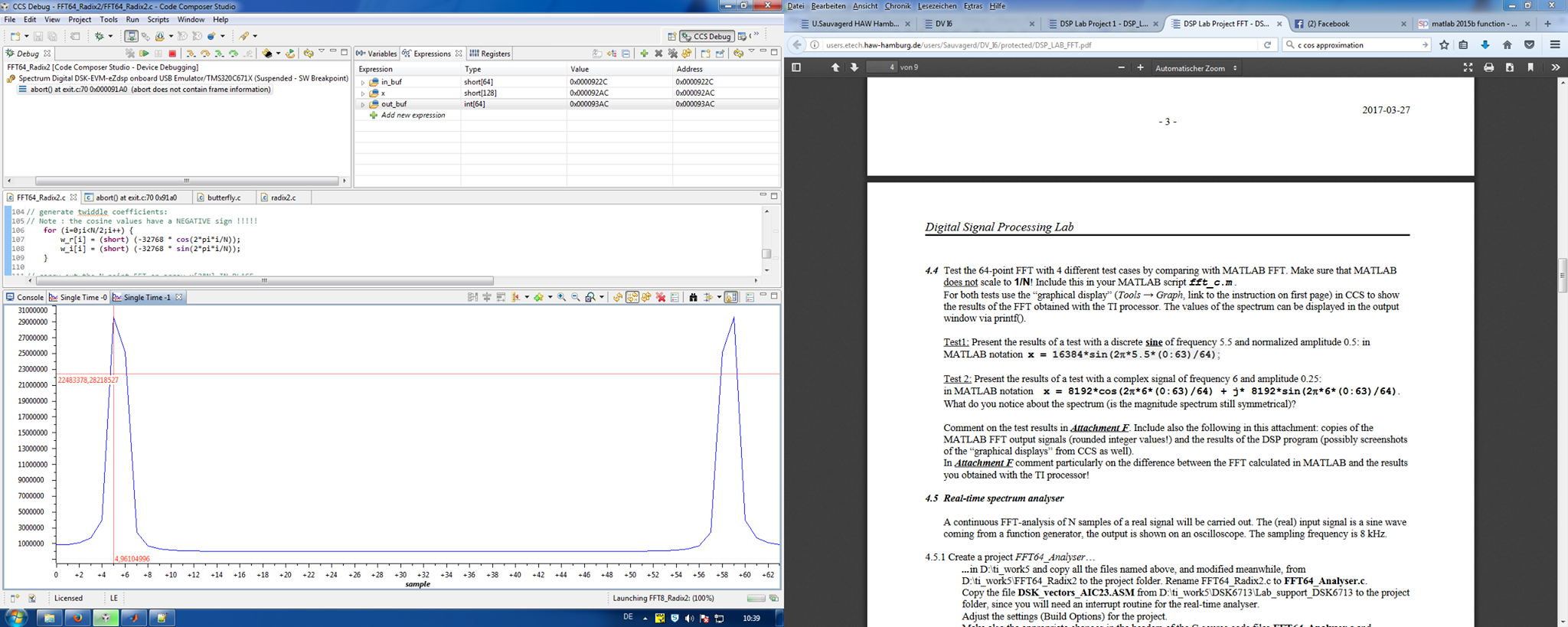


Figure – DSP board, magnitude of the 64 points FFT of input (1)

The buffer screenshot above shows the resulting FFT. Since we chose a frequency of 5.5 the peak is not correctly sampled because we only have values for i=5 and i=6. This explains the odd shape of the peaks.

The result in MATLAB is consistent with the previousoutput:

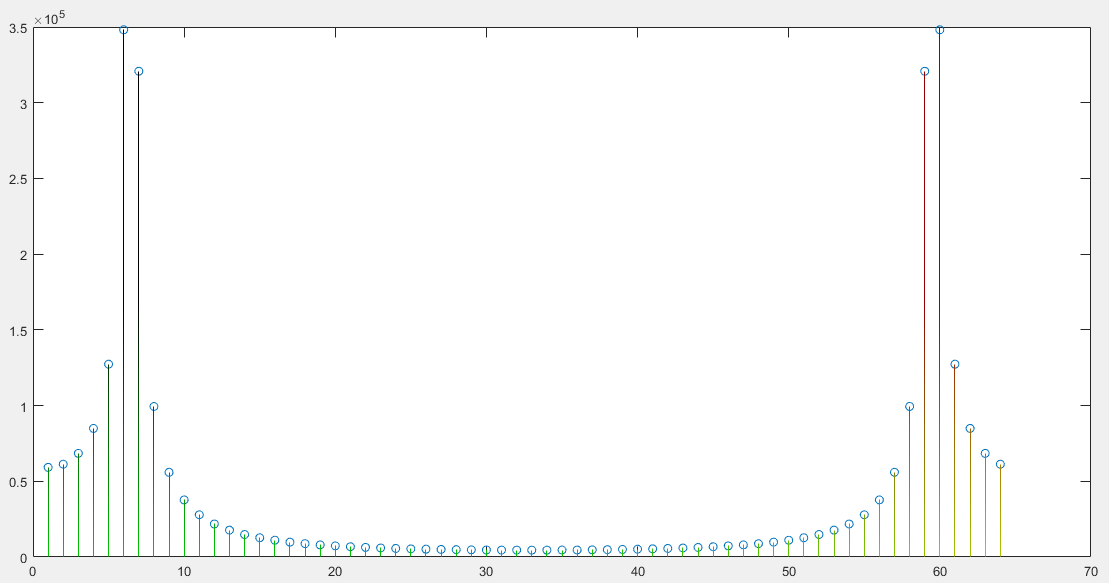


Figure – MATLAB, magnitude of the 64 points FFT of input (1)

|  |  |
| --- | --- |
| k | |X(k)| |
| 0 | 59204 |
| 1 | 61278 |
| 2 | 68449 |
| 3 | 84900 |
| 4 | 127331 |
| 5 | 348166 |
| 6 | 320769 |
| 7 | 99424 |
| 8 | 55922 |
| 9 | 37715 |
| 10 | 27876 |
| 11 | 21803 |
| 12 | 17731 |
| 13 | 14844 |
| 14 | 12711 |
| 15 | 11087 |
| 16 | 9820 |
| 17 | 8813 |
| 18 | 8000 |
| 19 | 7337 |
| 20 | 6791 |
| 21 | 6337 |
| 22 | 5960 |
| 23 | 5645 |
| 24 | 5383 |
| 25 | 5165 |
| 26 | 4986 |
| 27 | 4842 |
| 28 | 4728 |
| 29 | 4642 |
| 30 | 4581 |
| 31 | 4546 |
| 32 | 4534 |
| 33 | 4546 |
| 34 | 4581 |
| 35 | 4642 |
| 36 | 4728 |
| 37 | 4842 |
| 38 | 4986 |
| 39 | 5165 |
| 40 | 5383 |
| 41 | 5645 |
| 42 | 5960 |
| 43 | 6337 |
| 44 | 6791 |
| 45 | 7337 |
| 46 | 8000 |
| 47 | 8813 |
| 48 | 9820 |
| 49 | 11087 |
| 50 | 12711 |
| 51 | 14844 |
| 52 | 17731 |
| 53 | 21803 |
| 54 | 27876 |
| 55 | 37715 |
| 56 | 55922 |
| 57 | 99424 |
| 58 | 320769 |
| 59 | 348166 |
| 60 | 127331 |
| 61 | 84900 |
| 62 | 68449 |
| 63 | 61278 |

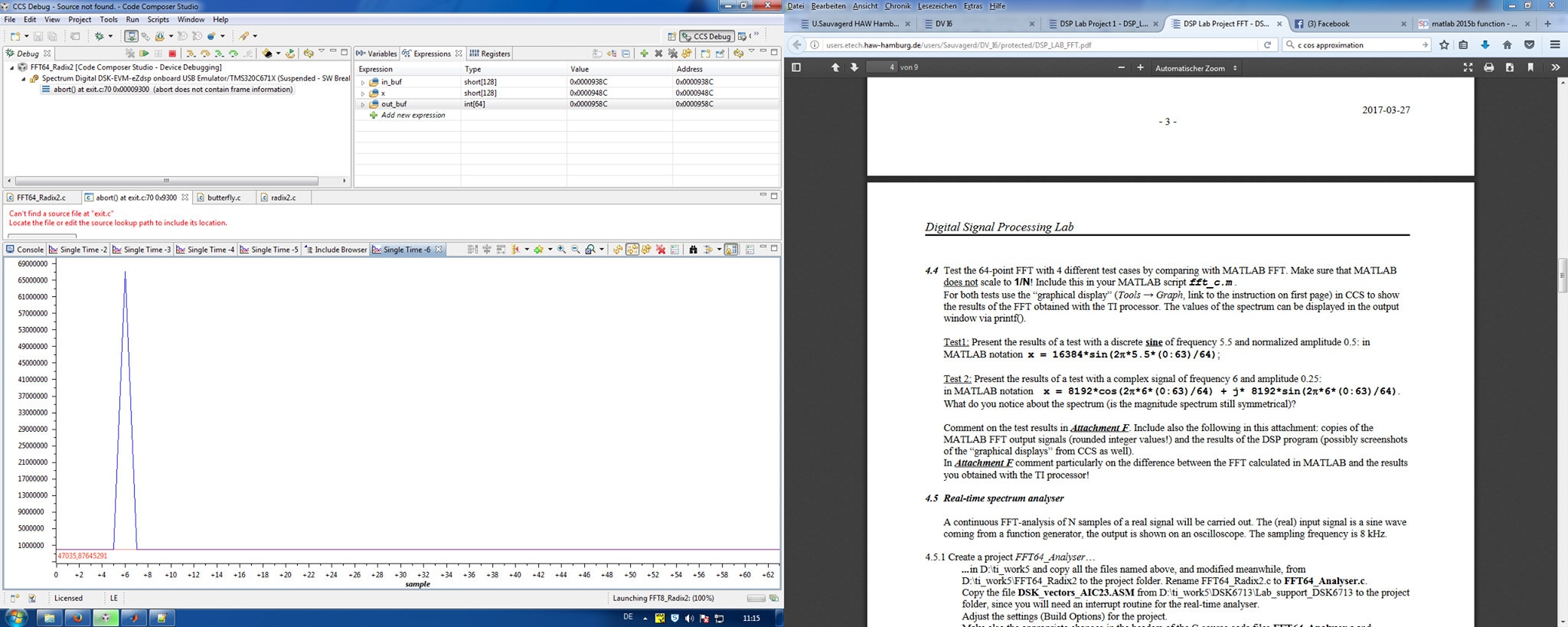


Figure – DSP board, magnitude of the 64 points FFT of input (2)

When computing the frequency spectrum of a complex signal, it is important to denote that the real and the imaginary parts are superimposed. Since the spectrum of a sine wave incorporates a peak with negative amplitude, this peak cancels its cosine counterpart at that frequency, thus only one peak remains as observable above. In this case, we sample the frequency correctly since the frequency is a multiple of .

Once again, the result in MATLAB are consistent:

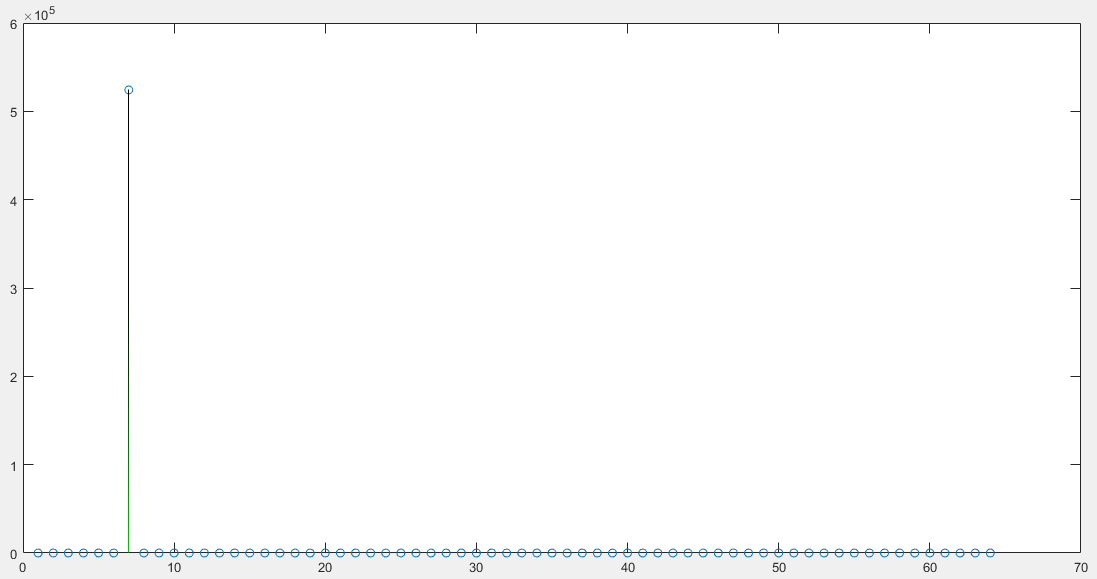


Figure - MATLAB, magnitude of the 64 points FFT of input (2)

|  |  |
| --- | --- |
| k | |X(k)| |
| 0 | 0 |
| 1 | 0 |
| 2 | 0 |
| 3 | 0 |
| 4 | 0 |
| 5 | 0 |
| 6 | 524288 |
| 7 | 0 |
| 8 | 0 |
| 9 | 0 |
| 10 | 0 |
| 11 | 0 |
| 12 | 0 |
| 13 | 0 |
| 14 | 0 |
| 15 | 0 |
| 16 | 0 |
| 17 | 0 |
| 18 | 0 |
| 19 | 0 |
| 20 | 0 |
| 21 | 0 |
| 22 | 0 |
| 23 | 0 |
| 24 | 0 |
| 25 | 0 |
| 26 | 0 |
| 27 | 0 |
| 28 | 0 |
| 29 | 0 |
| 30 | 0 |
| 31 | 0 |
| 32 | 0 |
| 33 | 0 |
| 34 | 0 |
| 35 | 0 |
| 36 | 0 |
| 37 | 0 |
| 38 | 0 |
| 39 | 0 |
| 40 | 0 |
| 41 | 0 |
| 42 | 0 |
| 43 | 0 |
| 44 | 0 |
| 45 | 0 |
| 46 | 0 |
| 47 | 0 |
| 48 | 0 |
| 49 | 0 |
| 50 | 0 |
| 51 | 0 |
| 52 | 0 |
| 53 | 0 |
| 54 | 0 |
| 55 | 0 |
| 56 | 0 |
| 57 | 0 |
| 58 | 0 |
| 59 | 0 |
| 60 | 0 |
| 61 | 0 |
| 62 | 0 |
| 63 | 0 |

# Attachment G

As discussed with you in the lecture, we had some trouble capturing a suitable screenshot during the lab so we decided to repeat this part at the next lab session.