ADwin

FFT (Fast Fourier Transform) Library

for ADwin Systems with ADSP processor (T9, T10)

Version 4.01

January 2004

Important hint :

- For the reason of correct working of this library, it is neccessary to have at least ADbasic Version 3.20.01 or newer.

Table of contents

FFT

Syntax: `return_value = FFT(real, img, result_real, result_img, help_array1, help_array2, number)`

parameter:	type:	size :	description :
real:	float array	number	real part of the original data
img:	float array	number	imaginary part of the original data
result_real:	float array	number*4	real part of the Fourier transformed data
result_img:	float array	number*4	imaginary part of the Fourier transformed data
help_array1:	float array	number*4	(memory for internal calculations)
help_array2:	float array	number*4	(memory for internal calculations)
number:	long		number of points for the FFT (must be a multiple of the number 2)
return_value:	void		(no return value)

Does a Fast Fourier Transform for a complex dataarray.

- The result of the FFT is not scaled onto the size of the components of the original data. If a scaling is necessary, there has to be called in addition the function *FFT_scale()* .
- Though the transformed data are stored in arrays of the size number*4 , the valid data are located in the array- elements 1 to number/2. (The other elements are used for internal calculations).
- Scaling (normalization) of the frequency- axis :
  If the original data was made up of a sampled signal with the total sampling time of "T_total" , it is possible to scale the frequency- axis by :
  The first element of the transformed data- arrays corresponds to the frequency 0 Hz.
  The last valid element (at position number/2) corresponds to the frequency
  f = (*number* / 2 – 1) / T_total
  The i -th element corresponds to the frequency = (i – 1) / T_total
  (Remark : The factor (i – 1) is caused by the fact, that arrays are starting always with element 1, not with element 0).
  In the example below the element 1024 corresponds to a frequency of (1024 – 1) / 0.1 s = 10230 Hz.
- For the reason of getting correct results, the frequency components of the original data should be within the following range : fmin = samplingfrequency / *number*
  fmax = samplingfrequency / 2 (because „Aliasing")

Example (for ADwin- GOLD or L16) :

Sample file :
C:\ADwin\ADbasic\lib\FFT_doc+demo**FFT_demo.bas**

Description :
If a sine-wave signal of 1000 Hz is connected to the analog input 1, there will be found a maximum each at DATA_3[101] and DATA_4[101] .

Hint:

- If *number* will not be modified on multiple calls of *FFT()* , the needed CPU- time can be reduced by using the functions *FFT_init()* and *FFT_calc()* instead of *FFT()* .

Hint to all functions of the FFT- library :

- Declaring the arrays in the internal memory (“DM_LOCAL“) reduces the amount of the needed CPU- time (ca. 23 ms compared to ca. 35 ms at a 1024 point FFT for T9 processor).
- It is necessary, that the FFT- library functions are called either in a low priority process or in the FINISH section or in the LOWINIT section (FINISH and LOWINIT sections are also executed with low priority). Otherwise the FFT would interrupt / block the communication to the ADBasic system for a too long time, causing an error.

FFT_mag

Syntax: *return_value = FFT_mag(real, img, result_mag, number2)*

parameter:	type:	size :	description :
real:	float array	number	real part of the original data
img:	float array	number	imaginary part of the original data
result_mag:	float array	number2	result : magnitude of the complex data
number2:	long		number of the data elements to be converted
return_value:	void		(no return value)

Converts a complex dataarray into an array, which contains the magnitudes of the complex elements.

- The magnitude of each element is calculated with the formula : result_mag[i] = SQRT(real[i]^2 + img[i]^2)
- Because the result of the function *FFT()* is a complex spectrum, this function *FFT_mag()* is an alternative way of displaying the result of the FFT() as a magnitude function (optionally together with the phase function *FFT_phase()*) .
- If this function is applied to the result of the function *FFT()* , *number2* should be = number_FFT_points / 2 .

Example (for ADwin- GOLD and L16) :

Sample file :
C:\ADwin\ADbasic\lib\FFT_doc+demo**FFT_mag_demo.bas**

Description :
If a sine-wave signal of 1500 Hz is connected to the analog input 1, there will be found a maximum at DATA_5[151] .

Hint to all functions of the FFT- library :

- Declaring the arrays in the internal memory (“DM_LOCAL“) reduces the amount of the needed CPU- time (ca. 23 ms compared to ca. 35 ms at a 1024 point FFT for T9 processor).
- It is necessary, that the FFT- library functions are called either in a low priority process or in the FINISH section or in the LOWINIT section (FINISH and LOWINIT sections are also executed with low priority). Otherwise the FFT would interrupt / block the communication to the ADBasic system for a too long time, causing an error.

FFT_scale

Syntax: *return_value = FFT_scale(unscaled, result_scaled, number2)*

parameter:	type:	size :	description :
unscaled:	float array	number2	(unscaled) data
result_scaled:	float array	number2	result: scaled data
number2:	long		number of the data elements to be scaled (must be = number_FFT_points / 2)
return_value:	void		(no return value)

Scales the result of a FFT, because the function *FFT()* does not scale to the size of the components of the original data.

- This function is based on the formula :
  For i <> 1 : result_scaled[i] = unscaled[i] / number2
  For i = 1 : result_scaled[1] = unscaled[1] / (number2 * 2)
- *number2* **must be** = number_FFT_points / 2 .
- This function does NOT scale (normalize) the frequency- axis of the spectrums (please see the comments to the function *FFT()*).

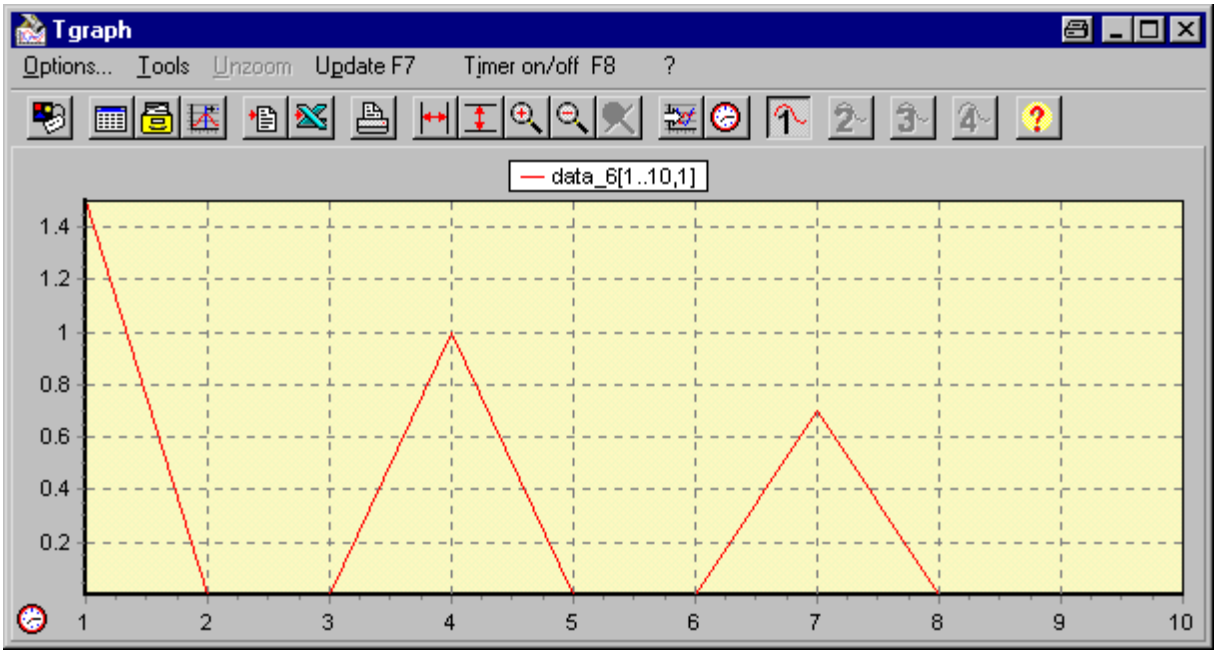
Example (for all ADwin- systems with T9 or T10 processor) :

Sample file :
C:\ADwin\ADbasic\lib\FFT_doc+demo**FFT_scale_demo.bas**

Description :
The program generates internally a frequency of 60 Hz with the amplitude 0.7 (unitless). Onto this frequency another frequency of 30 Hz with the amplitude 1.0 and a constant offset of 1.5 is overlayed. After calling the functions *FFT(), FFT_mag()* and *FFT_scale()* we get the following spectrum :

DATA_6[7] = 1	(60 Hz)
DATA_6[4] = 0.7	(30 Hz)
DATA_6[1] = 1.5	(constant offset)
DATA_6[all other elements] = 0	

Remark : All amplitudes of the shown frequencies correspond exactly to the sizes of the components of the original data.



Hint to all functions of the FFT- library :

- Declaring the arrays in the internal memory (“DM_LOCAL“) reduces the amount of the needed CPU- time (ca. 23 ms compared to ca. 35 ms at a 1024 point FFT for T9 processor).
- It is necessary, that the FFT- library functions are called either in a low priority process or in the FINISH section or in the LOWINIT section (FINISH and LOWINIT sections are also executed with low priority). Otherwise the FFT would interrupt / block the communication to the ADBasic system for a too long time, causing an error.

FFT_phase

Syntax: *return_value = FFT_phase(real, img, result_phase, number2)*

parameter:	type:	size :	description :
real:	float array	number2	real part of the complex data
img:	float array	number2	imaginary part of the complex data
result_phase:	float array	number2	result : phase of the complex data
number2:	long		number of the data elements to be converted
return_value:	void		(no return value)

Converts a complex dataarray into an array, which contains the phase of the complex elements.

- The phase of each element is calculated with the formula :
  If real[i] >= 0 : result_phase[i] = ARCTAN(img[i] / real[i])
  If real[i] < 0 : result_phase[i] = ARCTAN(img[i] / real[i]) + *PI*
  (A detailled listing of this formula can be found in the file : math.inc)
- Because the result of the function *FFT()* is a complex spectrum, this function *FFT_phase()* is an alternative way of displaying the result of the FFT() as a phase function (optionally together with the magnitude function *FFT_mag()*) .
- If this function is applied to the result of the function *FFT()* , *number2* should be = number_FFT_points / 2 .

Example (for all ADwin- systems with T9 or T10 processor) :

Sample file :
C:\ADwin\ADbasic\lib\FFT_doc+demo**FFT_phase_demo.bas**

Description :
This program generates internally two frequencies with 30 Hz. The second frequency has a phase displacement of *PI* / 2 . Both signals are sampled synchronously. After calling the functions *FFT(), FFT_mag(), FFT_scale()* and *FFT_phase()* separately for each signal, we get spectrums with the following characteristics :

DATA_6[4] = 1	(30 Hz)
DATA_6[all other elements] = 0	
DATA_7[4] = -0.018410	(phase = ca. 0)
DATA_26[4] = 1	(30 Hz)
DATA_26[all other elements] = 0	
DATA_27[4] = 1.552389	(phase = ca. *PI* / 2)

Remark : The phase of all other elements is NOT defined (i.e. is probably not 0).

Hint to all functions of the FFT- library :

- Declaring the arrays in the internal memory (“DM_LOCAL“) reduces the amount of the needed CPU- time (ca. 23 ms compared to ca. 35 ms at a 1024 point FFT for T9 processor).
- It is necessary, that the FFT- library functions are called either in a low priority process or in the FINISH section or in the LOWINIT section (FINISH and LOWINIT sections are also executed with low priority). Otherwise the FFT would interrupt / block the communication to the ADBasic system for a too long time, causing an error.

FFT_mag_scale

Syntax: `return_value = FFT_mag_scale(real, img, result_m_s, number2)`

parameter:	type:	size :	description :
`real`:	float array	number	real part of the original data
`img`:	float array	number	imaginary part of the original data
`result_m_s`:	float array	number2	result : scaled magnitude of the complex data
`number2`:	long		number of the data elements to be converted
`return_value`:	void		(no return value)

This function is a combination out of ***FFT_mag()*** and ***FFT_scale()*** and therefore a little bit faster than calling those two functions.

Example (for all ADwin- systems with T9 or T10 processor) :

Sample file :
C:\ADwin\ADbasic\lib\FFT_doc+demo**FFT_scale_demo_opt.bas**

FFT_init

Syntax: `return_value = FFT_init( help_array1, help_array2, number)`

parameter:	type:	size :	description :
`help_array1`:	float array	number*4	(memory for internal calculations)
`help_array2`:	float array	number*4	(memory for internal calculations)
`number`:	long		number of points for the FFT (must be a multiple of the number 2)
`return_value`:	void		(no return value)

Only necessary and reasonable together with a succeeding call of *FFT_calc()* or *FFT_calc_DM()*, etc.
The function *FFT()* consists of the two functions *FFT_init()* and *FFT_calc().*
FFT_init() is reasonable, if the Fast Fourier Transform has to be performed several times. Then *FFT_init()* has to be called just for one time.
But, the parameter *number* has to have the same value on all calls of *FFT_init()* and *FFT_calc().*

Example (for all ADwin- systems with T9 or T10 processor) :

Sample file :
C:\ADwin\ADbasic\lib\FFT_doc+demo**FFT_scale_demo_opt.bas**

FFT_calc

Syntax: `return_value = FFT_calc(real, img, result_real, result_img, help_array1, help_array2, number)`

parameter:	type:	size :	description :
`real`:	float array	number	real part of the original data
`img`:	float array	number	imaginary part of the original data
`result_real`:	float array	number*4	real part of the Fourier transformed data
`result_img`:	float array	number*4	imaginary part of the Fourier transformed data
`help_array1`:	float array	number*4	(memory for internal calculations)
`help_array2`:	float array	number*4	(memory for internal calculations)
`number`:	long		number of points for the FFT (must be a multiple of the number 2)
`return_value`:	void		(no return value)

Only reasonable together with a preceeding call of *FFT_init()*.
The function *FFT()* consists of the two functions *FFT_init()* and *FFT_calc().*
FFT_calc() is reasonable, if the Fast Fourier Transform has to be performed several times. Then *FFT_init()* has to be called just for one time.

Example (for all ADwin- systems with T9 or T10 processor) :

Sample file :
C:\ADwin\ADbasic\lib\FFT_doc+demo**FFT_scale_demo_opt.bas**

FFT_calc_DM

Syntax: `return_value = FFT_calc_DM(real, img, result_real, result_img, help_array1, help_array2, number)`

parameter:	type:	size :	description :
`real`:	float array	number	real part of the original data
`img`:	float array	number	imaginary part of the original data
`result_real`:	float array	number*4	real part of the Fourier transformed data
`result_img`:	float array	number*4	imaginary part of the Fourier transformed data
`help_array1`:	float array	number*4	(memory for internal calculations)
`help_array2`:	float array	number*4	(memory for internal calculations)
`number`:	long		number of points for the FFT (must be a multiple of the number 2)
`return_value`:	void		(no return value)

Only reasonable together with a preceeding call of *FFT_init()*.
The function *FFT()* consists of the two functions *FFT_init()* and *FFT_calc()*.
FFT_calc()_DM is a special version of *FFT_calc()*, which is optimized for the T10 processor. It can be used also for the T9 processor, but there will be no optimization effect with the T9.
FFT_calc()_DM may only be used, if the arrays are declared in the internal memory ("DM_LOCAL").
The needed CPU- time is ca. 11 ms at a 1024 point FFT for the T10 processor, compared to ca. 14 ms with *FFT_calc()* (Remark: BOTH time measurements done with arrays in DM_LOCAL).

Example (for all ADwin- systems with T9 or T10 processor) :

Sample file :
C:\ADwin\ADbasic\lib\FFT_doc+demo**FFT_scale_demo_opt.bas**

FFT_calc_DX

Syntax: `return_value = FFT_calc_DX(real, img, result_real, result_img, help_array1, help_array2, number)`

parameter:	type:	size :	description :
`real`:	float array	number	real part of the original data
`img`:	float array	number	imaginary part of the original data
`result_real`:	float array	number*4	real part of the Fourier transformed data
`result_img`:	float array	number*4	imaginary part of the Fourier transformed data
`help_array1`:	float array	number*4	(memory for internal calculations)
`help_array2`:	float array	number*4	(memory for internal calculations)
`number`:	long		number of points for the FFT (must be a multiple of the number 2)
`return_value`:	void		(no return value)

Only reasonable together with a preceeding call of *FFT_init()*.
The function *FFT()* consists of the two functions *FFT_init()* and *FFT_calc()*.
FFT_calc_DX() is a special version of *FFT_calc()*, which is optimized for the T10 processor. It can be used also for the T9 processor, but there will be no optimization effect with the T9. *FFT_calc_DX()* may only be used, if the arrays are declared in the external memory ("DRAM_EXTERN") (= default).
The needed CPU- time is ca. 49 ms at a 2048 point FFT for the T10 processor, compared to ca. 53 ms with *FFT_calc()* (Remark: BOTH time measurements done with arrays in DRAM_EXTERN).

Example (for all ADwin- systems with T9 or T10 processor) :

Sample file :
C:\ADwin\ADbasic\lib\FFT_doc+demo**FFT_scale_demo_opt_DX.bas**