# UM0585
# User manual

## STM32F10x
## DSP library

## Introduction

This user manual describes the STM32F10x DSP (digital signal processing) library, which is a suite of common digital signal processing functions:

- PID controller
- Fast Fourier transform
- FIR and IIR filters

The library contains C and assembly functions. The assembly code is ported on ARM®, GCC and IAR Systems™ assemblers.

# Contents

# List of tables

# List of figures

# 1 DSP Library description

The STM32F10x DSPLib is a suite of common functions for signal processing. It includes the following functions:

**Table 1. STM32F10x DSP library functions**

| Function name | Description |
|---|---|
| `DoPID` | PID controller in C, error computed outside the routine |
| `DoFullPID` | PID controller in C, error computed inside the routine |
| `PID_stm32` | PID controller in ASM, error computed outside the routine |
| `cr4_fft_64_stm32` | Complex radix-4 FFT for a 16-bit, 64-point vector |
| `cr4_fft_256_stm32` | Complex radix-4 FFT for a 16-bit, 256-point vector |
| `cr4_fft_1024_stm32` | Complex radix-4 FFT for a 16-bit, 1024-point vector |
| `fir_16by16_stm32` | 16-bit FIR filter |
| `iiarma_stm32` | 16-bit, 4th-order IIR filter |
| `iir_biquad_stm32` | 16-bit, 8th-order, biquad IIR filter |

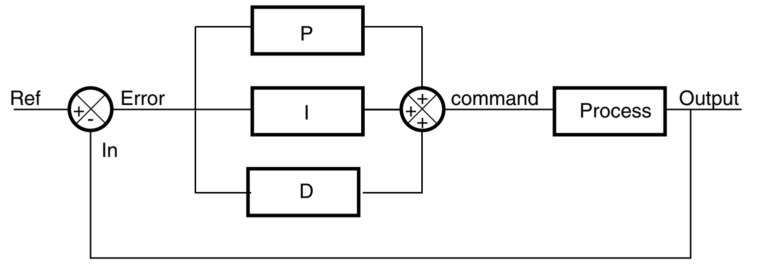
# 2 PID controller

## 2.1 Description

The proportional-integral-derivative or PID controller is commonly used in the industry. It is a feedback loop controller that manages the process command with a view to reducing the error between the desired set point and the measured process variable.

The following block diagram shows the parallel structure of a PID controller. This is the structure implemented in this DSP library.

**Figure 1. Block diagram of PID controller**



## 2.2 DSP library functions

The DSP library provides three PID functions:

- **`DoPID`**: a PID core loop coded in C (the error is computed outside the function)
- **`DoFullPID`**: a full PID controller coded in C (with error computing)
- **`PID_stm32`**: an optimized PID core loop written in assembly

### 2.2.1 `DoPID` function

*Table 2* describes the `DoPID` function.

**Table 2. `DoPID` function**

| Function name | `DoPID` |
|---|---|
| Prototype | `uint16_t DoPID(uint16_t Error, uint16_t *Coeff)` |
| Behavior description | PID in C, error computed outside the function. |
| Input parameter | – Error: difference between reference and measured values<br>– Coeff: pointer to the coefficients table |
| Output parameter | None |
| Return parameter | PID output command |

#### Example

```
/* Fill the coefficients table */
Coeff[0] = Kp;/*proportional coefficient*/
```

```
Coeff[1] = Ki;/*integral coefficient*/
Coeff[2] = Kd;/*derivative coefficient*/
/* Compute the error */
Error = Target_Signal - Measured_Signal;
/* PID control process */
Command = DoPID(Error, Coeff);
```

### 2.2.2 `DoFullPID` function

*Table 3* describes the `DoFullPID` function.

**Table 3. `DoFullPID` function**

| Function name | `DoFullPID` |
|---|---|
| Prototype | `uint16_t DoFullPID(uint16_t In, uint16_t Ref, uint16_t *Coeff)` |
| Behavior description | PID in C, error computed inside the function. |
| Input parameter | – In: Input (measured value)<br>– Ref: reference (target value)<br>– Coeff: pointer to the coefficients table |
| Output parameter | Computed error |
| Return parameter | PID output command |

#### Example

```
/* Fill the coefficients table */
Coeff[0] = Kp;/*proportional coefficient*/
Coeff[1] = Ki;/*integral coefficient*/
Coeff[2] = Kd;/*derivative coefficient*/

/* PID control process, the error is computed inside the function */
Command = DoFullPID(Measured_Signal, Target_Signal, Coeff);
```

### 2.2.3 `PID_stm32` function

*Table 4* describes the `PID_stm32` function.

**Table 4. `PID_stm32` function**

| Function name | `PID_stm32` |
|---|---|
| Prototype | `uint16_t PID_stm32(uint16_t Error, uint16_t *Coeff);` |
| Behavior description | Assembly optimized PID controller with error computed outside the function. |
| Input parameter | – Error: difference between reference and measured values<br>– Coeff: pointer to the coefficients table |
| Output parameter | None |
| Return parameter | PID output command |

The `PID_stm32` function is used in the same way as the `DoPID` function. The error must be computed, then the `PID_stm32` function is called to improve the PID control process and to return the appropriate command according to the coefficients table.

# 3 Complex 16-bit radix-4 FFT

## 3.1 Description

The discrete Fourier transform (DFT) converts N complex values from the time domain to the frequency domain.

The fast Fourier transform (FFT) is an optimized algorithm designed to compute the DFT efficiently.

The STM32F10x DSP library provides a complex radix-4, with decimation-in-time, linear-order FFT.

Let x[N] be the time signal samples. To use the FFT functions of the DSP library, the following conditions must be satisfied:

- N is a power of 4
- All the signal samples must be 32-bit data containing the 16-bit real part followed by the 16-bit imaginary part (in the little Endian order: imaginary_real).

## 3.2 DSP library functions

The DSP provides three complex 16-bit radix-4 FFT functions:

1. **`cr4_fft_64_stm32`**: an optimized FFT function to compute 64-point DFT
2. **`cr4_fft_256_stm32`**: an optimized FFT function to compute 256-point DFT
3. **`cr4_fft_1024_stm32`**: an optimized FFT function to compute 1024-point DFT

### 3.2.1 `cr4_fft_64_stm32` function

*Table 5* describes the `cr4_fft_stm32` function.

**Table 5. `cr4_fft_64_stm32` function**

| Function name | **`cr4_fft_64_stm32`** |
|---|---|
| Prototype | `void cr4_fft_64_stm32(void *pssOUT, void *pssIN, uint16_t Nbin);` |
| Behavior description | complex 16-bit, 64-point radix-4 FFT |
| Input parameter | – `pssOUT`: pointer to the output array data<br>– `pssIN`: pointer to the input array data<br>– `Nbin`: the number of points, must be 64. |
| Output parameter | None |
| Return parameter | None |

### 3.2.2 `cr4_fft_256_stm32` function

*Table 6* describes the `cr4_fft_256_stm32` function.

**Table 6. `cr4_fft_256_stm32` function**

| Function name | cr4_fft_256_stm32 |
|---|---|
| Prototype | `void cr4_fft_256_stm32(void *pssOUT, void *pssIN, uint16_t Nbin);` |
| Behavior description | complex 16-bit, 256-point radix-4 FFT |
| Input parameter | – `pssOUT`: pointer to the output array data<br>– `pssIN`: pointer to the input array data<br>– `Nbin`: the number of points, must be 256. |
| Output parameter | None |
| Return parameter | None |

### 3.2.3 `cr4_fft_1024_stm32` function

*Table 7* describes the `cr4_fft_1024_stm32` function.

**Table 7. `cr4_fft_1024_stm32` function**

| Function name | cr4_fft_1024_stm32 |
|---|---|
| Prototype | `void cr4_fft_1024_stm32(void *pssOUT, void *pssIN, uint16_t Nbin);` |
| Behavior description | complex 16-bit, 1024-point radix-4 FFT |
| Input parameter | – `pssOUT`: pointer to the output array data<br>– `pssIN`: pointer to the input array data<br>– `Nbin`: the number of points, must be 1024. |
| Output parameter | None |
| Return parameter | None |

#### Example

```
#define N 64 /*Number of points*/
uint32_t x[N],y[N]; /* input and output arrays */
uint16_t real[N], imag[N]; /* real and imaginary arrays */
/* Fill the input array */
for(i=0; i<N; i++)
  x[i] = (((uint16_t)(real[i])) | ((uint32_t)(imag[i]<<16)));
cr4_fft_64_stm32(y, x, N); /*computes the FFT of the x[N] samples*/
```

## 3.3 FFT performance improvement

The FFT coefficients table is stored in the Flash memory since it is declared as code in the assembly file. The performance of the FFT function can be improved by placing the FFT coefficients in RAM. This is done as described below:

1. Comment all the FFT coefficients in the FFT function assembly file.

```
/* TableFFT_V7
          ;N=16
         DC16 0x4000,0x0000, 0x4000,0x0000, 0x4000,0x0000
         DC16 0xdd5d,0x3b21, 0x22a3,0x187e, 0x0000,0x2d41
         DC16 0xa57e,0x2d41, 0x0000,0x2d41, 0xc000,0x4000
         DC16 0xdd5d,0xe782, 0xdd5d,0x3b21, 0xa57e,0x2d41
         ; N=64
         DC16 0x4000,0x0000, 0x4000,0x0000, 0x4000,0x0000
         DC16 0x2aaa,0x1294, 0x396b,0x0646, 0x3249,0x0c7c
        ...
*/
```

2. Then, in the main, include the *table_fft.h* file, which is a part of the DSP library.
3. Finally, go to the FFT function assembly code, and inverse the comment in the following lines:

```
ADRL    R0, TableFFT_V7    /* Coeff in Flash */
//LDR.W R0, =TableFFT        /* Coeff in RAM */
```

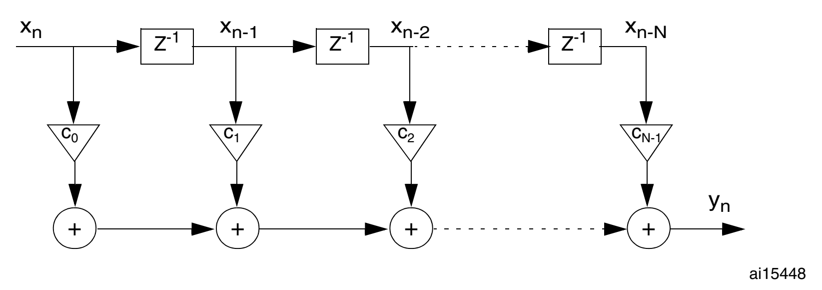
# 4 16-bit FIR filter

## 4.1 Description

The finite impulse filter (FIR) is a digital filter that is linearly dependent on a fixed finite number of input samples.

The FIR filter can be defined by the number of coefficients to be processed (also called taps), which gives the number of MAC (multiply-accumulate) operations to be done.

**Figure 2. Block diagram of an FIR filter of length N**



The FIR filter of the DSP library is a direct-form real FIR filter that uses an array of M 16-bit coefficients to filter N 16-bit samples.

Let us put:

- a, the output vector of length N
- c, the coefficients vector of length M
- x, the input vector

So, x must have a length of M + N – 1.

## 4.2 DSP library function

The FIR function of the DSP library is an optimized assembly function that takes into account the load-store architecture of the Cortex™-M3. Therefore, and as an optimization constraint, the number of taps and the number of output samples must be a multiple of 4.

### 4.2.1 `fir_16by16_stm32` function

*Table 8* describes the `fir_16by16_stm32` function.

**Table 8. `fir_16by16_stm32` function**

| Function name | `fir_16by16_stm32` |
|---|---|
| Prototype | `void fir_16by16_stm32(int *a,short *x,struct COEFS *p,unsigned int N)` |
| Behavior description | Block Fir 16-bit filter. |
| Input parameter | – a: output array<br>– x: input array<br>– p: pointer to the coefficient structure of the COEFS type<br>– N: number of output samples |
| Output parameter | None |
| Return parameter | None |

The filter coefficients and their number must be filled into a structure of the COEFS type.

The coefficients structure is defined as follows:

```
typedef struct {
  short *h;
  unsigned int nh;
}COEFS;
```

### Example

```
#define M 32 /*number of coefficients*/
#define N 32 /*number of output samples*/

COEFS fir_coefs;/*coefficients structure*/

int a[N];/*filter output vector*/
short x[M+N-1] = {x_0,x_1...,x_{M+N-1}};/*filter input vector*/
short h[M]={h_0,h_1...,h_{M-1}};/*filter coefficients vector*/

fir_coefs.nh = M; /*Number of Coefficients for FIR*/
fir_coefs.h = h; /*Pointer on FIR coefficient vector*/

fir_16by16_stm32(a,x,&fir_coefs,N);/*performs the FIR filtering*/
```

# 5 16-bit IIR filters

## 5.1 Description

The infinite impulse response (IIR) filter is a digital filter that depends linearly on a finite number of input samples and a finite number of previous filter outputs.

The IIR filter is represented by *Equation 1* below.

**Equation 1**

$$y[n] = \sum_{i=0}^{M} b_i x[n-i] - \sum_{i=1}^{N} a_i y[n-i]$$

*Equation 1* is known as an auto-regressive moving average form (ARMA). The first sum of *Equation 1* represents the moving average part, which is similar to an FIR block, and the second sum represents the auto-regressive part, which is the feedback from previous outputs.

This IIR filter structure is called direct form I.

*Figure 3* shows the structure of the direct form I of a second-order IIR filter. Direct form I uses four delays to realize a second-order IIR filter.

To reduce the number of delays, the canonical form may be used. *Figure 4* represents the block diagram of the canonical form of a second-order IIR filter. The number of delays is reduced from 4 to only 2.

Figure 3. Block diagram of the direct form I of second-order IIR filter

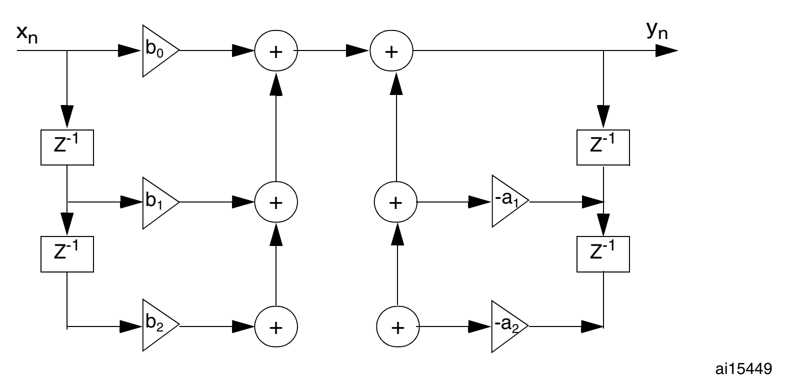
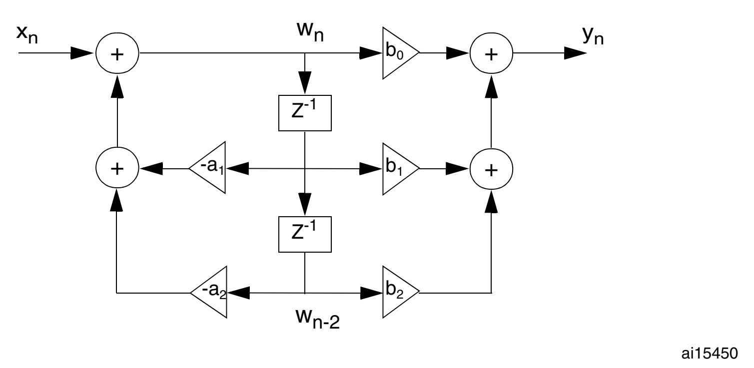


**Figure 4. Block diagram of the canonical form of a second-order IIR filter**



The DSP library implements the two forms (direct form I and the canonical form) of the IIR filter:

- Direct form I is used to design an IIR filter of order 4 (ARMA IIR)
- The canonical form is used to design an IIR filter of order 8, by using 4 second-order IIR filter sections (biquads) arranged in series (biquad IIR filter).

## 5.2 DSP library functions

The DSP library provides two IIR filters:

1. `iiarma_stm32`: an ARMA IIR filter, designed with 4 auto-regressive and 5 moving-average filter coefficients, so, in *Equation 1*, M = 4 and N = 4. The function code is written in assembly.
2. `iir_biquad_stm32`: a biquad IIR filter, designed by connecting 4 biquads in series. The function code is written in C.

### 5.2.1 `iiarma_stm32` function

*Table 9* describes the `iirarma_stm32` function.

**Table 9. `iirarma_stm32` function**

| Function name | `iirarma_stm32` |
|---|---|
| Prototype | `void iirarma_stm32(short *y, short *x, short *h2, short *h1, int ny)` |
| Behavior description | 16-bit, auto-regressive moving-average IIR (ARMA) filter. |
| Input parameter | – y: output array of length ny+4<br>– x: input array of length ny+4<br>– h2: filter coefficient vector, moving-average part.<br>– h1: filter coefficient vector, auto-regressive part.<br>– ny: number of output samples |
| Output parameter | None |
| Return parameter | None |

To use the `iirarma_stm32` function, the following conditions must be satisfied:

- The moving-average coefficient vector, h2, must have a length of 5 shorts:
  - `short h2[5];`
- The auto-regressive coefficient vector, h1, must have a length of 5 shorts. The h1[0] value is not used, so 4 coefficients remain:
  - `short h1[5];`
- The number of output samples, ny, must be a multiple of 4 greater than or equal to 8.
- Input and output vectors must have a length of (ny+4) shorts.
- The first four elements of the output vector must have the previous outputs.

#### Example

```
#define NY 32/*number of outputs, must be a multiple of 4 and >= 8*/
/* Coefficients for the ARMA IIR filter */
short h2[5] = { 0x09c2,  0x270a,  0x3a8f,  0x270a,  0x09c2 };
short h1[5] = { 0x7fff,  0xd24a,  0x72ca,  0xcf4e,  0x1ad4 };
/* Input and output vectors */
short x[NY+4],y[NY+4];

/* Fill the input vector x */
...
/* Fill the 4 previous outputs */
y[0] = y0;
y[1] = y1;
y[2] = y2;
y[3] = y3;
/* Improve the filtering of NY samples */
iirarma_stm32(y, x, h2, h1, NY);
```

### 5.2.2 `iir_biquad_stm32` function

*Table 10* describes the `iir_biquad_stm32` function.

**Table 10. `iir_biquad_stm32` function**

| Function name | iir_biquad_stm32 |
|---|---|
| Prototype | `void iir_biquad_stm32(uint16_t *y, uint16_t *x, int16_t *IIRCoeff, uint16_t ny)` |
| Behavior description | 8th-order 4 biquad IIR filter |
| Input parameter | – y: output array<br>– x: input array<br>– IIRCoeff: IIR filter coefficients<br>– ny: number of output samples |
| Output parameter | None |
| Return parameter | None |

### Example

```
#define NY 32/*number of outputs*/
/* Coefficients for the biquad IIR filter: 4 sections, with 5
coefficients in each section */
int16_t Coeff[20] = {...};
/* Input and output vectors */
short x[NY],y[NY];
/* Fill the input vector x */
...
/* Improve the filtering of NY samples */
iir_biquad_stm32(y, x, Coeff, NY);
```

# 6 STM32F10x DSP library benchmark

This section provides the STM32F10x DSP library benchmark results, which are computed using the IAR EWARM 5.20 toolchain, with high-speed optimization.

## 6.1 Function code footprint

Table 11. STM32F10x DSP library functions code footprint

| Function name | Code size (bytes) |
|---|---|
| `DoPID` | 52 |
| `DoFullPID` | 58 |
| `PID_stm32` | 72 |
| `cr4_fft_64_stm32` | 718(1) |
| `cr4_fft_256_stm32` | 1486(1) |
| `cr4_fft_1024_stm32` | 4560(1) |
| `fir_16by16_stm32` | 162 |
| `iiarma_stm32` | 156 |
| `iir_biquad_stm32` | 294 |

1. FFT code size was computed with FFT coefficients table stored in Flash memory. If the FFT coefficients are stored in SRAM, the code size of the three FFT functions is equal to 480 bytes.

## 6.2 Function execution time

### 6.2.1 PID controller

Table 12. PID controller, error computed outside the routine

| PID | 24 MHz 0 wait state | | 48 MHz 1 wait state | | 72 MHz 2 wait states | |
|---|---|---|---|---|---|---|
| | cycle count | time | cycle count | time | cycle count | time |
| ASM function | 45 | 1.87 µs | 51 | 1.06 µs | 59 | 0.819 µs |
| C function | 47 | 1.96 µs | 50 | 1.04 µs | 54 | 0.75 µs |

Analysis of the PID timing shows that assembly code is not as fast as C code. The compiler is more efficient in accessing variables than manual optimization (offset computation and data placement in literal pool).

Table 13. PID controller, error computed within the routine

| PID | 24 MHz 0 wait state | | 48 MHz 1 wait state | | 72 MHz 2 wait states | |
|---|---|---|---|---|---|---|
| | cycle count | time | cycle count | time | cycle count | time |
| C function | 48 | 2 µs | 52 | 1.08 µs | 57 | 0.79 µs |

### 6.2.2 Fast Fourier transform (FFT)

Table 14. Complex radix 4, 16-bit FFT, coefficients in Flash memory

| FFT (ASM funct.) | 24 MHz 0 wait state | | 48 MHz 1 wait state | | 72 MHz 2 wait states | |
|---|---|---|---|---|---|---|
| | cycle count | time | cycle count | time | cycle count | time |
| 64 points | 3847 | 0.16 ms | 4 472 | 0.093 ms | 5 661 | 0.078 ms |
| 256 points | 21 039 | 0.876 ms | 24 964 | 0.52 ms | 31 527 | 0.437 ms |
| 1024 points | 100 180 | 4.174 ms | 114 350 | 2.382 ms | 153 930 | 2.138 ms |

Table 15. Complex radix 4, 16-bit FFT, coefficients in RAM

| FFT (ASM funct.) | 24 MHz 0 wait state | | 48 MHz 1 wait state | | 72 MHz 2 wait states | |
|---|---|---|---|---|---|---|
| | cycle count | time | cycle count | time | cycle count | time |
| 64 points | 3 847 | 0.16 ms | 4 025 | 0.084 ms | 4 764 | 0.066 ms |
| 256 points | 21 039 | 0.876 ms | 22 176 | 0.462 ms | 26 065 | 0.362 ms |
| 1024 points | 100 180 | 4.174 ms | 102 057 | 2.126 ms | 127 318 | 1.768 ms |

### 6.2.3 FIR filter

Table 16. 16-bit, 32-tap FIR filter

| FIR (ASM filter) | 24 MHz 0 wait state | | 48 MHz 1 wait state | | 72 MHz 2 wait states | |
|---|---|---|---|---|---|---|
| | cycle count | time | cycle count | time | cycle count | time |
| 32 samples | 3516 | 146.5 µs | 3525 | 73.4 µs | 3727 | 51.76 µs |

### 6.2.4 IIR filters

Table 17. 16-bit canonic form, 4 biquad IIR filter

| IIR (C filter) | 24 MHz 0 wait state | | 48 MHz 1 wait state | | 72 MHz 2 wait states | |
|---|---|---|---|---|---|---|
| | cycle count | time | cycle count | time | cycle count | time |
| 32 samples | 3478 | 144.9 µs | 3636 | 75.75 µs | 3929 | 54.57 µs |

Table 18. 16-bit, 4th-order IIR filter

| IIR (ASM filter) | 24 MHz 0 wait state | | 48 MHz 1 wait state | | 72 MHz 2 wait states | |
|---|---|---|---|---|---|---|
| | cycle count | time | cycle count | time | cycle count | time |
| 32 samples | 1696 | 70 µs | 1761 | 36.69 µs | 1986 | 27.58 µs |

# 7 STM32F10x DSP demo description

The STM32F10x DSP demo is an example that illustrates how to use the 64-point FFT function of the STM32F10x DSP library. It consists of an LCD that displays the FFT transformation.

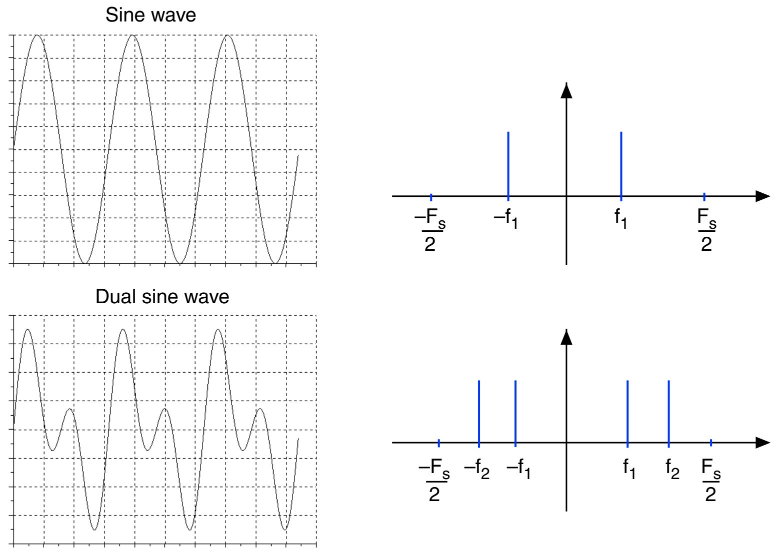
The demo runs on the STM3210B-EVAL board, and shows the 64-point FFT transformation of two signals:

1. a sine wave
2. a dual sine wave

The two types of wave are displayed at variable frequencies and for each frequency the FFT is computed and the result is plotted as shown in *Figure 5*:

- The frequency ($f_1$) of the sine wave is increased from 40 Hz to 4 kHz by steps of 30 Hz.
- The dual sine wave is the sum of a constant-frequency ($f_2$) sine wave and the above described variable-frequency ($f_1$) sine wave.

**Figure 5. FFT of a sine wave with frequency $f_1$ and of a dual sine wave with frequencies $f_1$ and $f_2$, both sampled at a frequency $F_s$**



# 8 Conclusion

This user manual describes the STM32F10x DSP library, which contains:

- a PID controller
- complex 16-bit radix-4 FFT optimized functions for 64, 256 and 1024 points
- a 16-bit FIR filter
- a 16-bit direct-form I IIR filter
- a 16-bit canonical-form IIR filter designed by biquads

# 9 Revision history

**Table 19. Document revision history**

| Date | Revision | Changes |
|---|---|---|
| 13-Oct-2008 | 1 | Initial release. |
| 04-Jun-2010 | 2 | *Introduction* modified.<br>*Table 1: STM32F10x DSP library functions* updated.<br>*Section 7: STM32F10x DSP demo description* added.<br>`u32` changed to `uint32_t`.<br>`u16` changed to `uint16_t`.<br>`s16` changed to `int16_t`.<br>Small text changes. |

**Please Read Carefully:**

Information in this document is provided solely in connection with ST products. STMicroelectronics NV and its subsidiaries (“ST”) reserve the right to make changes, corrections, modifications or improvements, to this document, and the products and services described herein at any time, without notice.

All ST products are sold pursuant to ST’s terms and conditions of sale.

Purchasers are solely responsible for the choice, selection and use of the ST products and services described herein, and ST assumes no liability whatsoever relating to the choice, selection or use of the ST products and services described herein.

No license, express or implied, by estoppel or otherwise, to any intellectual property rights is granted under this document. If any part of this document refers to any third party products or services it shall not be deemed a license grant by ST for the use of such third party products or services, or any intellectual property contained therein or considered as a warranty covering the use in any manner whatsoever of such third party products or services or any intellectual property contained therein.

**UNLESS OTHERWISE SET FORTH IN ST’S TERMS AND CONDITIONS OF SALE ST DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY WITH RESPECT TO THE USE AND/OR SALE OF ST PRODUCTS INCLUDING WITHOUT LIMITATION IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE (AND THEIR EQUIVALENTS UNDER THE LAWS OF ANY JURISDICTION), OR INFRINGEMENT OF ANY PATENT, COPYRIGHT OR OTHER INTELLECTUAL PROPERTY RIGHT.**

**UNLESS EXPRESSLY APPROVED IN WRITING BY AN AUTHORIZED ST REPRESENTATIVE, ST PRODUCTS ARE NOT RECOMMENDED, AUTHORIZED OR WARRANTED FOR USE IN MILITARY, AIR CRAFT, SPACE, LIFE SAVING, OR LIFE SUSTAINING APPLICATIONS, NOR IN PRODUCTS OR SYSTEMS WHERE FAILURE OR MALFUNCTION MAY RESULT IN PERSONAL INJURY, DEATH, OR SEVERE PROPERTY OR ENVIRONMENTAL DAMAGE. ST PRODUCTS WHICH ARE NOT SPECIFIED AS "AUTOMOTIVE GRADE" MAY ONLY BE USED IN AUTOMOTIVE APPLICATIONS AT USER’S OWN RISK.**

Resale of ST products with provisions different from the statements and/or technical features set forth in this document shall immediately void any warranty granted by ST for the ST product or service described herein and shall not create or extend in any manner whatsoever, any liability of ST.

ST and the ST logo are trademarks or registered trademarks of ST in various countries.

Information in this document supersedes and replaces all information previously supplied.

The ST logo is a registered trademark of STMicroelectronics. All other names are the property of their respective owners.

© 2010 STMicroelectronics - All rights reserved

STMicroelectronics group of companies

Australia - Belgium - Brazil - Canada - China - Czech Republic - Finland - France - Germany - Hong Kong - India - Israel - Italy - Japan - Malaysia - Malta - Morocco - Philippines - Singapore - Spain - Sweden - Switzerland - United Kingdom - United States of America

**www.st.com**