AN1444
APPLICATION NOTE

General Purpose DSP Library for the ST120 DSP

By David DAUBOIS

1 - INTRODUCTION

This document presents a set of optimized DSP functions for C programmers on the ST120 DSP. These functions are typically used in real time applications where execution time is critical.

Using this "ready to use" library can significantly decrease the development time in complex applications.

TABLE OF CONTENTS PAGE

2 - USING THE LIBRARY

2.1 - Warranty

The library package is distributed free of charge.There are no explicit guarantees .

2.2 - Library Content

The library package consists of:

– The object library: "genlib.a".
– A header file: "genlib.h".
– Two files including the tables of coefficients for the FFT algorithms: cR2FFTCoefs.c, cR2FFT3216Coefs.c
– The assembly source codes of the functions (to allow customizing).

2.3 - Calling a Function

The functions were written to be called by a C language source code. Calling from assembly language source is possible with respect with parameter passing (see description of EABI in [2]).

To include a function in a C language source code, it is needed to

– Include the "genlib.h" include file.
– Include the "st1types.h" include file (part of the C compiler package).
– Link the code with the object library file "genlib.a".

2.4 - Data Types

st1types.h (C compiler package) defines the different data types used for ST120 programming.

The following types are used in the library:

– fract16: 16-bit fractional type. The name of the function includes "q15"
– fract32: 32-bit fractional type. The name of the function includes "q31"
– int16: 16-bit integer type. The name of the function includes "16" or "i16"
– int32: 32-bit integer type. The name of the function includes "32" or "i32"

2.5 - Note on Cycle Measurement

The number of cycles is calculated for best case. It can be different as it depends on data placement (internal memories, external memories...). The measurement is made on simulator (st100serv) between the decode of the first instruction of the function and the decode of the first instruction following the return of the function. Measurement on the chip may slightly differ.

2.6 - Naming Convention and Parameters Passing

The name of a function begins with a lowercase. Each uppercase starts a new word in the name. Underscore is used to separate type of data. A type before the name indicates the type of the return value of the function. Types after the name indicate the types of the parameters passed to the function.

When applicable, the parameters are passed in the following order:

– Pointer on an output array
– Pointer on an input array
– Pointer on coefficients array
– Size of output array
– Size of input array
– Size of coefficient array
– Other integer parameters

2.7 - Bit scaling

Integer or fixed point data type are used in the library. To avoid overflow, it can be useful to divide the intermediate result by a value before continuing computation. It is obtained by shifting right the result by one or more bits. Each shift by one divides the result by two (but there is a loss of precision).

3 - LIBRARY FUNCTIONS

3.1 - Alphabetical List of LIBRARY FUNCTIONS

3.2 - aCorrScal_q15

aCorrScal_q15 (fract16 *resVec, fract16 *vec, int16 nR, int16 nI, int16 scale)

Description:

Auto correlation of a vector.

Arguments:

resVec	Pointer to result vector.
vec	Pointer to the input vector.
nR	Size of result vector.
nI	Size of input vector.
scale	Number of bit scaling for each multiply (see algorithm).

Algorithm:

$$resVec(j) = \sum_{k=0}^{nI-j-1} vec(j+k) \times vec(k)$$

Assembly source code:

aCorrScal_q15.s

Code size and cycles:

Size	276 bytes
Cycles	

3.3 - bitRev_16

bitRev_16 (int16 *vec, int16 nvec)

Description:

In place bit-reversing of a vector of nvec 16-bit words. This function is used in conjunction with a FFT function to reorder the input or the output.

Arguments:

vec	Pointer to input/output vector.
nvec	Number of elements in input vector (multiple of 4).

Note:

Can also be used with fractional type vector (q1.15 format).

Assembly source code:

bitRev_16.s

Code size and cycles:

Size	196 bytes
Cycles	7nvec/4 + 33

3.4 - bitRev_32

bitRev_32 (int32 *vec, int16 nvec)

Description:

In place bit-reversing of a vector of nvec 32-bit words. This function is used in conjunction with a FFT function to reorder the input or the output.

Arguments:

vec	Pointer to input/output vector.
nvec	Number of elements in input vector (multiple of 4).

Note:

Can also be used with fractional type vector (q1.31 format).

Assembly source code:

bitRev_31.s

Code size and cycles:

Size	140 bytes
Cycles	< 11nvec/4 + 24

3.5 - cFIR_q15

cFIR_q15 (fract16 *out, fract16 *in, fract16 *coefs, fract16 **delay, int16 nIn, int16 nCoefs, int16 scale)

Description:

Complex FIR filter (direct form) of a 16-bit fractional vector.

Arguments:

out	Pointer to output vector (Re0, Im0, Re1, Im1, Re2, Im2...).
in	Pointer to input vector (Re0, Im0, Re1, Im1, Re2, Im2...).
coef	Pointer to coefficient (Re(b0), Im(b0), Re(b1), Im(b1)...). Maximum of 16 complex coefficients.
delay	Pointer to a delay line (used for internal computation). Allows successive blocks filtering.
nIn	Number of complex elements in input vector (multiple of 2).
nCoefs	Number of complex coefficients.
scale	Number of bit scaling for each output.

Algorithm:

$$y(n) = \sum_{i=0}^{nCoefs-1} b_i \times x(n-i)$$

Notes:

Clamped multiply accumulate.

Maximum use of 16 complex coefficients due to constant size delay line.

Delay is used for internal computation. It must be define as a fract16 array of size 32 (16 Reals + 16 Imaginaries) and must be aligned on a 64- byte boundary.

Before the first call to the function, the delay line must be initialized to 0.

To optimize speed, delay must not be placed in the same memory bank as data and coefficients.

Example of passing delay line:

```
fract16 delay[32] = {0};
fract16 *ptrDelay;
...
void main()
{
ptrDelay = delay;
...
cFIR_q15 (...,...,...,&ptrDelay,...,...,...);
...
}
```

Assembly source code:

cFIR_q15.s

Code size and cycles:

Size	196 bytes
Cycles	< (2+2nCoefs)nIn+ 46

3.6 - conv_q15

conv_q15 (fract16 *y, fract16 *x, fract16 *h, int16 nx, int16 nh)

Description:

Full length convolution between vector x and h.

Arguments:

y	Pointer to result vector (size nx+nh-1).
x	Pointer to the input vector1 (size nx).
h	Pointer to input vector2 (size nh).
nx	Size of input vector x.
nh	Size of input vector h.

Algorithm:

$$y(j) = \sum_{k=0}^{nh-1} h(k) \times x(j-k)$$

Assembly source code:

conv_q15.s

Code size and cycles:

Size	148 bytes
Cycles	nh(3nh+3nx-1)+2nx+27

3.7 - conv2_q15

conv2_q15 (fract16 *y, fract16 *x, fract16 *h, int16 ny, int16 nh)

Description:

Convolution between vector x and h.

Arguments:

y	Pointer to result vector (size ny). Must be a multiple of 2.
x	Pointer to the input vector1 (size ny+nh-1).
h	Pointer to input vector2 (size nh).
ny	Size of output vector y.
nh	Size of input vector h.

Algorithm:

$$y(j) = \sum_{k=0}^{nh-1} h(k) \times x(j-k)$$

Notes:

The nh-1 first values of x correspond to negative values of index (j-k<0).

To optimize speed, h must not be placed in the same bank of memory than x and y.

Assembly source code:

conv2_q15.s

Code size and cycles:

Size	164 bytes
Cycles	ny/2(nh+1)+29

3.8 - cR2FFT_q15_q15

cR2FFT_q15_q15 (fract16 *outR, fract16 *outI, fract16 *inR, fract16 *inI, fract16 *coefOd, fract16 *coefEv, int16 n)

Description:

Radix 2 complex Fast Fourier Transform (decimation in frequency) with integrated bit-reverse (no needs to bit-reverse the data). 16-bit fractional coefficients, 16-bit fractional data.

Arguments:

outR	Pointer to real output vector. Must be aligned on a FFT size boundary. Must be placed in X memory for optimized performance.
outI	Pointer to Imaginary output vector. Must be aligned on a FFT size boundary. Must be placed in Y memory for optimized performance.
inR	Pointer to real input vector. Must be placed in X memory for optimized performance.
inI	Pointer to Imaginary input vector. Must be placed in Y memory for optimized performance.
coefOd	Pointer to odd coefficients vector (see file cR2FFTCoefs.c). Must be placed in X memory for optimized performance.
coefOEv	Pointer to even coefficients vector (see file cR2FFTCoefs.c). Must be placed in Y memory for optimized performance.
n	Power of two corresponding to the size of the FFT. n = Log(size)/Log(2). 4 <= n <= 10

Algorithm:

$$X(k) = \sum_{n=0}^{N-1} x(n) \times e^{\frac{-j2\pi nk}{N}}$$

Notes:

At the end of the computation, the content of the input vector is lost.

The cR2FFTCoefs.c file includes coefficients needed for FFT with a size between 16 and 1024.

A value must be defined in the cR2FFTCoefs.c file to choose the size.

Example of use for FFT 1024:

```
In file cR2FFTCoefs.c:
...
//#define FFT512
#define FFT1024                                  // choose FFT 1024 coefficients

In main file:
#include "genlib.h"
...
fract16 inR[1024];                               // real input
fract16 inI[1024];                               // imaginary input
// align the following array on a 1024 boundary (see documentation of
// the C compiler package)
fract16 outR[1024];                              // real output
// align the following array on a 1024 boundary (see documentation of
// the C compiler package)
fract16 outI[1024];                              // imaginary output
extern fract16 coefOd[];                         // coefOd is defined in the file cR2FFTCoefs.c
extern fract16 coefOd[];                         // coefOd is defined in the file cR2FFTCoefs.c
...
void main()
{
// Fill the input arrays
...
cR2FFT_q15_q15(outR, outI, inR, inI, coefOd, coefEv, 10);
...
}
```

Assembly source code:

cR2FFT_q15_q15.s

Code size and cycles:

Size 760 bytes (without coefficient vectors)

Cycles:

FFT	Cycles
FFT16	224
FFT32	372
FFT64	692
FFT128	1404
FFT256	2996
FFT512	6540
FFT1024	14372

3.9 - cR2FFT_q31_q15

cR2FFT_q31_q15 (fract32 *dataR, fract32 *dataI, fract16 *coefs, int16 n)

Description:

In place radix 2 complex Fast Fourier Transform (decimation in frequency). 16-bit fractional coefficients, 32-bit fractional data.

Arguments:

dataR	Pointer to real input/output vector.
dataI	Pointer to imaginary input/output vector. Must be placed in another memory bank for optimized performance (X memory if Y is default).
coefs	Pointer to coefficients vector (see cR2FFT3216Coefs.c file).
n	Power of two corresponding to the size of the FFT. n = Log(size)/Log(2). 4 <= n <= 10.

Algorithm:

$$X(k) = \sum_{n=0}^{N-1} x(n) \times e^{\frac{-j2\pi nk}{N}}$$

Notes:

The cR2FFT3216Coefs.c file includes coefficients for FFT with a size between 16 and 1024.

A value must be defined in the cR2FFT3216Coefs.c file to choose the size.

The result values are bit-reversed (use bitRev_32 function on real and imaginary vector to obtain "in order" data).

Example of use for FFT 1024:

In file cR2FFT3216Coefs.c:

```
...
//#define FFT512
#define FFT1024                                  // choose the FFT 1024 coefficients

In main file:
#include "genlib.h"
...
fract32 dataR[1024];                             // real input/output
fract32 dataI[1024];                             // imaginary input/output
extern fract16 coefs[];                          // coefs is defined in the file cR2FFT3216Coefs.c
...
void main()
{
// Fill the input arrays
...
cR2FFT_q31_q15(dataR,dataI,coefs,10);
bitRev_32(dataR,1024);                           // Real data are reordered.
bitRev_32(dataI,1024);                           // Imaginary data are reordered.
...
}
```

Assembly source code:

cR2FFT_q31_q15.s

Code size and cycles:

Size 756 bytes (without coefficient vectors)

Cycles:

FTT	Cycles
FFT16	247
FFT32	523
FFT64	1167
FFT128	2659
FFT256	6071
FFT512	13771
FFT1024	30943

3.10 - FIR_q15

FIR_q15 (fract16 *out, fract16 *in, fract16 *coefs, fract16 **delay, int16 nIn, int16 nCoefs, int16 scale)

Description:

Real FIR filter (direct form) of a 16-bit fractional vector.

Arguments:

out	Pointer to output vector.
in	Pointer to input vector.
coefs	Pointer to coefficient (b0, b1, b2 ..). Maximum of 16 coefficients.
delay	Pointer to a delay line (used for internal computation). Allow successive blocks filtering.
nIn	Number of elements in input vector (multiple of 2).
nCoefs	Number of coefficients (maximum 32).
scale	Number of bit scaling for each output.

Algorithm:

$$y(n) = \sum_{i=0}^{nCoefs-1} b_i \times x(n-i)$$

Notes:

Clamped multiply accumulate.

Maximum use of 32 coefficients due to fixed size delay line.

Delay is used for internal computation. It must be defined as a fract16 array of size 64 and must be aligned on a 128 bytes boundary.

Before the first call to the function, the delay line must be initialized to 0.

To optimize speed, Delay must not be placed in the same memory bank as data and coefficients.

Example of passing Delay line:

```
fract16 delay[64] = {0};
fract16 *ptrDelay;
...
void main()
{
ptrDelay = delay;
...
FIR_q15 (...,...,...,&ptrDelay,...,...,...);
...
}
```

Assembly source code:

FIR_q15.s

Code size and cycles:

Size 164 bytes

Cycles (5+nCoefs/2)nIn/2+ 48

3.11 - i16_maxValGP16_16

int16 r = i16_maxValGP16_16(int16 *i, int16 ni)

Description:

Return the maximum element of an integer or fractional 16-bit vector.

Arguments:

i	Pointer to input vector.
ni	Number of elements in input vector. 1 <= ni <= 32767.
r	Maximum value in the vector.

Notes:

Use of GP16 instruction set for code density.

Can be used with fractional type vector.

Assembly source code:

i16_maxValGP16_16.s

Code size and cycles:

Size	30 bytes
Cycles	18ni + 25

3.12 - i16_maxValSLIW_16

int16 r = i16_maxValSLIW_16(int16 *i, int16 ni)

Description:

Return the maximum element of an integer or fractional 16-bit vector.

Arguments:

i	Pointer to input vector.
ni	Number of elements in input vector. 1 <= ni <= 32767.
r	Maximum value in the vector.

Notes:

Use of SLIW instruction mode for high performance DSP oriented code.

Can be used with fractional type vector.

Assembly source code:

i16_maxValSLIW_16.s

Code size and cycles:

Size	120 bytes
Cycles	ni/2 + 25

3.13 - i32_maxValGP16_32

int32 r = i32_maxValGP16_32(int32 *i, int16 ni)

Description:

Return the maximum element of an integer or fractional 32-bit vector.

Arguments:

i	Pointer to input vector.
ni	Number of elements in input vector. 1 <= ni <= 32767.
r	Maximum value in the vector.

Notes:

Use of GP16 instruction set for code density.

Can be used with fractional type vector.

Assembly source code:

i32_maxValGP16_32.s

Code size and cycles:

Size	30 bytes
Cycles	18ni + 25

3.14 - i32_maxValSLIW_32

int32 r = i32_maxValSLIW_32(int32 *i, int16 ni)

Description:

Return the maximum element of an integer or fractional 32-bit vector.

Arguments:

i	Pointer to input vector.
ni	Number of elements in input vector. 1 <= ni <= 32767.
r	Maximum value in the vector.

Notes:

Use of SLIW instruction for high performance DSP oriented code.

Can be used with fractional type vector.

Assembly source code:

i32_maxValSLIW_32.s

Code size and cycles:

Size	108 bytes
Cycles	ni + 23

3.15 - i16_minValGP16_16

int16 r = i16_minValGP16_16(int16 *i, int16 ni)

Description:

Return the minimum element of an integer or fractional 16-bit vector.

Arguments:

i	Pointer to input vector.
ni	Number of elements in input vector. 1 <= ni <= 32767.
r	Minimum value in the vector.

Notes:

Use of GP16 instruction set for code density.

Can be used with fractional type vector.

Assembly source code:

i16_minValGP16_16.s

Code size and cycles:

Size	32 bytes
Cycles	20ni + 24

3.16 - i16_minValSLIW_16

int16 r = i16_minValSLIW_16(int16 *i, int16 ni)

Description:

Return the minimum element of an integer or fractional 16-bit vector.

Arguments:

i	Pointer to input vector.
ni	Number of elements in input vector. 1 <= ni <= 32767.
r	Minimum value in the vector.

Notes:

Use of SLIW instruction for high performance DSP oriented code.

Can be used with fractional type vector.

Assembly source code:

i16_minValSLIW_16.s

Code size and cycles:

Size	120 bytes
Cycles	ni/2 + 22

3.17 - i32_minValGP16_32

int32 r = i32_minValGP16_32(int32 *i, int16 ni)

Description:

Return the minimum element of an integer or fractional 32-bit vector.

Arguments:

i	Pointer to input vector.
ni	Number of elements in input vector. 1 <= ni <= 32767.
r	Minimum value in the vector.

Notes:

Use of GP16 instruction set for code density.

Can be used with fractional type vector.

Assembly source code:

i32_minValGP16_32.s

Code size and cycles:

Size	34 bytes
Cycles	19ni + 25

3.18 - i32_minValSLIW_32

int32 r = i32_minValSLIW_32(int32 *i, int16 ni)

Description:

Return the minimum element of an integer or fractional 32-bit vector.

Arguments:

i	Pointer to input vector.
ni	Number of elements in input vector. 1 <= ni <= 32767.
r	Minimum value in the vector.

Notes:

Use of SLIW instruction mode to get the fastest execution code.

Can be used with fractional type vector.

Assembly source code:

i32_minValSLIW_32.s

Code size and cycles:

Size	108 bytes
Cycles	ni + 23

3.19 - IIRBiquad4_q15

IIRBiquad4_q15 (fract16 *out, fract16 *in, fract16 *coefs, fract16 *delay, int16 n, int16 nb)

Description:

IIR real filter with n cascaded biquads of 4 coefficients.

Arguments:

out	Pointer to output vector (size n).
in	Pointer to input vector (size n).
coefs	Pointer to coefficients (-b1i, a1i, -b2i, a2i...) (size 4*nb).
delay	Pointer to a delay line (size 2*nb).
n	Number of element in input and output vectors.
nb	Number of biquads.

Algorithm:

General form for one biquad:

y(n) = x(n) + a1x(n-1) + a2x(n-2) - b1y(n-1) - b2y(n-2)

Direct form II uses for computation (for one biquad):

d(n) = x(n) - b1d(n-1) - b2 d(n-2)

y(n) = d(n) + a1d(n-1) + a2d(n-2)

d is the delay line.

Notes:

The results are clamped to the maximum q1.15 value.

Before the first call to the function, the delay line must be initialized to 0.

To optimize speed, delay must not be placed in the same bank of memory as in and coefs.

Assembly source code:

IIRBiquad4_q15.s

Code size and cycles:

Size	148 bytes
Cycles	< n(4nb+2)+27 (for nb >=3)

3.20 - mMul_16_16_16

mMul_16_16_16(int16 *matRes, int16 *mat1, int16 *mat2, int16 nr1, int16 nc2, int16 nc1)

Description:

Matrix multiply 16-bit integer, result on 16-bit integer (mat1 * mat2 -> matRes).

Arguments:

matRes	Pointer to result matrix.
mat1	Pointer to matrix 1.
mat2	Pointer to matrix 2.
nr1	Number of rows in matrix 1.
nc1	Number of columns in matrix 1.
nc2	Number of columns in matrix 2.

Algorithm:

$$matRes(i,j) = \sum_{k=1}^{p} mat1(i,k) \times mat2(k,j)$$

Notes:

No overflow handling.

To optimize speed, mat2 must not be placed in the same memory bank as mat1 and matRes.

Assembly source code:

mMul_16_16_16.s

Code size and cycles:

Size	244 bytes
Cycles	68 + nr1(1 + nc2(8 + 2(nr2/2 - 2))

3.21 - mMul_16_16_32

mMul_16_16_32(int32 *matRes, int16 *mat1, int16 *mat2, int16 nr1, int16 nc2, int16 nc1)

Description:

Matrix multiply 16-bit integer, result on 32-bit integer (mat1 * mat2 -> matRes).

Arguments:

matRes	Pointer to result matrix.
mat1	Pointer to matrix 1.
mat2	Pointer to matrix 2.
nr1	Number of rows in matrix 1.
nc1	Number of columns in matrix 1.
nc2	Number of columns in matrix 2.

Algorithm:

$$matRes(i,j) = \sum_{k=1}^{p} mat1(i,k) \times mat2(k,j)$$

Notes:

No overflow handling.

To optimize speed, mat2 must be placed in the other memory bank.

Assembly source code:

mMul_16_16_32.s

Code size and cycles:

Size	244 bytes
Cycles	68 + nr1(1 + nc2(8 + 2(nr2/2 - 2)))

3.22 - mMul_q15_q15_q15

mMul_q15_q15_q15(fract16 *matRes, fract16 *mat1, fract16 *mat2, int16 nr1, int16 nc2, int16 nc1)

Description:

Matrix multiply 16-bit fractional, result on 16-bit fractional (mat1 * mat2 -> matRes).

Arguments:

matRes	Pointer to result matrix.
mat1	Pointer to matrix 1.
mat2	Pointer to matrix 2.
nr1	Number of rows in matrix 1.
nc1	Number of columns in matrix 1.
nc2	Number of columns in matrix 2.

Algorithm:

$$matRes(i,j) = \sum_{k=1}^{p} mat1(i,k) \times mat2(k,j)$$

Notes:

No overflow handling.

To optimize speed, mat2 must be placed in the other memory bank.

Assembly source code:

mMul_q15_q15_q15.s

Code size and cycles:

Size	244 bytes
Cycles	68 + nr1(1 + nc2(8 + 2(nr2/2 - 2)))

3.23 - mMul_q15_q15_q31

mMul_q15_q15_q31(fract31 *matRes, fract16 *mat1, fract16 *mat2, int16 nr1, int16 nc2, int16 nc1)

Description:

Matrix multiply fractional 16-bit, result on fractional 32-bit (mat1 * mat2 -> matRes).

Arguments:

matRes	Pointer to result matrix.
mat1	Pointer to matrix 1.
mat2	Pointer to matrix 2.
nr1	Number of rows in matrix 1.
nc1	Number of columns in matrix 1.
nc2	Number of columns in matrix 2.

Algorithm:

$$matRes(i,j) = \sum_{k=1}^{p} mat1(i,k) \times mat2(k,j)$$

Notes:

No overflow handling.

To optimize speed, mat2 must be placed in the other memory bank.

Assembly source code:

mMul_q15_q15_q31.s

Code size and cycles:

Size	244 bytes
Cycles	68 + nr1(1 + nc2(8 + 2(nr2/2 - 2)))

3.24 - mVM_8_16_16

mVM_8_16_16(int16 *vecRes, char *mat, int16 *vec, int16 matR, int16 matC)

Description:

Matrix vector multiply (mat(8 bits integer) * vec(16-bit integer) -> vecRes(16-bit integer)).

Arguments:

vecRes	Pointer to result vector.
mat	Pointer to matrix.
vec	Pointer to vector.
matR	Number of rows in matrix.
matC	Number of columns in matrix 2.

Algorithm:

$$vecRes(i) = \sum_{k=1}^{j} mat(i,k) \times vec(k)$$

Notes:

The data are clamped during the sum.

To optimize speed, mat must be placed in the other memory bank.

Assembly source code:

mVM_8_16_16.s

Code size and cycles:

Size	132 bytes
Cycles	44 + matR(matC/2 + 2) (if matC>=6 and matR>=2)

3.25 - mVM_q7_q15_q15

mVM_q7_q15_q15(fract16 *vecRes, fract8 *mat, fract16 *vec, int16 matR, int16 matC)

Description:

Matrix vector multiply (mat(8 bits fractional) * vec(16-bit fractional) -> vecRes(16-bit fractional)).

Arguments:

vecRes	Pointer to result vector.
mat	Pointer to matrix.
vec	Pointer to vector.
matR	Number of rows in matrix.
matC	Number of columns in matrix 2.

Algorithm:

$$vecRes(i) = \sum_{k=1}^{j} mat(i,k) \times vec(k)$$

Notes:

The data are clamped during the sum.

To optimize speed, mat must be placed in the other memory bank.

Assembly source code:

mVM_q7_q15_q15.s

Code size and cycles:

Size	132 bytes
Cycles	44 + matR(matC/2 + 2) (if matC>=6 and matR>=2)

3.26 - q31_energy_q15

fract32 res = q31_energy_q15 (fract16 *vec, int16 size)

Description:

Sum of square of a 16-bit fractional vector. Result is 32-bit fractional.

Arguments:

res	Result.
vec	Pointer to the vector.
size	Size of vectors (multiple of 2).

Algorithm:

$$res = \sum_{i=1}^{size} vec1(i)^2$$

Note:

Each addition is clamped.

Assembly source code:

q31_energy_q15.s

Code size and cycles:

Size	84 bytes
Cycles	22 + (size-2)/2

3.27 - q31_vMul_q15

fract32 res = q31_vMul_q15 (fract16 *vec1,fract16 *vec2, int16 size)

Description:

Vector multiply (vec1(16-bit fractional) x vec2(16-bit fractional) -> res(32-bit fractional)).

Arguments:

res	Result.
vec1	Pointer to vector 1.
vec2	Pointer to vector 2.
size	Size of vectors (multiple of 2).

Algorithm:

$$res = \sum_{i=1}^{size} vec1(i) \times vec2(i)$$

Notes:

Each addition is clamped.

To optimize speed, vec2 must not be placed in the same memory bank as vec1.

Assembly source code:

q31_vMul_q15.s

Code size and cycles:

Size	52 bytes
Cycles	21 + size

3.28 - vAdd_16

vAdd_16(int16 *vecRes, int16 *vec1, int16 *vec2, int16 size)

Description:

Vector addition (integer or fractional): vec1(16-bit) + vec2(16-bit) -> vecRes(16-bit).

Arguments:

vecRes	Pointer to result vector.
vec1	Pointer to vector 1.
vec2	Pointer to vector 2.
size	Size of vectors (multiple of 2).

Algorithm:

$$vecRes(i) = vec1(i) + vec2(i)$$

Notes:

The data are clamped during the sum.

To optimize speed, vec2 must not be placed in the same memory bank as vec2 and vecRes.

Can be used with fractional type vectors.

Assembly source code:

vAdd_16.s

Code size and cycles:

Size	52 bytes
Cycles	21 + size

4 - LIBRARY ASSEMBLY SOURCES

This chapter includes the library assembly sources.Those programs are free of charge and there are no explicit guarantees.

4.1 - Alphabetical List of Library Assembly Sources

The following table lists the functions alphabetically starting from aCorrScal_q15.

4.2 - genLib.h

```
/*-----------------------------------------------------------------
* Project component : ST100 GENLIB
* File Name : genLib.h
* Purpose : General purpose DSP library for ST120
*-----------------------------------------------------------------
*/

#ifndef _GENLIB_H
#define _GENLIB_H

#include "st1types.h"
typedef char fract8;


int16 i16_minValGP16_16(int16 *, int16);
int16 i16_minValSLIW_16(int16 *, int16);
int32 i32_minValGP16_32(int32 *, int16);
int32 i32_minValSLIW_32(int32 *, int16);
int16 i16_maxValGP16_16(int16 *, int16);
int16 i16_maxValSLIW_16(int16 *, int16);
int32 i32_maxValGP16_32(int32 *, int16);
int32 i32_maxValSLIW_32(int32 *, int16);
void mMul_16_16_16 (int16 *, int16 *, int16 *, int16,int16,int16);
void mMul_16_16_32 (int32 *, int16 *, int16 *, int16,int16,int16);
void mMul_q15_q15_q15 (fract16 *, fract16 *, fract16 *, int16,int16,int16);
void mMul_q15_q15_q31 (fract32 *, fract16 *, fract16 *, int16,int16,int16);
void mVM_q7_q15_q15 (fract16 *, fract8 *, fract16 *, int16, int16);
void mVM_8_16_16 (int16 *, char *, int16 *, int16, int16);
void vAdd_16 (int16 *, int16 *,int16 *, int16);
fract32 q31_vMul_q15(fract16 *, fract16 *, int16);
fract32 q31_energy_q15(fract16 *, int16);
void bitRev_16(int16 *, int16);
void bitRev_32(int32 *, int16);
void FIR_q15(fract16 *, fract16 *, fract16 *, fract16 **, int16, int16, int16);
void cFIR_q15(fract16 *, fract16 *, fract16 *, fract16 **, int16, int16, int16);
void aCorrScal_q15(fract16 *, fract16 *, int16, int16, int16);
void cR2FFT_q15_q15(fract16 *,fract16 *, fract16 *, fract16 *, fract16 *, fract16 *,
int16);
void cR2FFT_q31_q15(fract32 *,fract32 *, fract16 *, int16);
void IIRBiquad4_q15(fract16 *, fract16 *, fract16 *, fract16 *, int16 , int16);
void conv_q15(fract16 *, fract16 *, fract16 *, fract16 , fract16 );
void conv2_q15(fract16 *, fract16 *, fract16 *, fract16 , fract16 );
#endif /* _GENLIB_H */
```

4.3 - cR2FFTCoefs.c

```
/*-------------------------------------------------------------
* Project component : ST100 GENLIB
* File Name : cR2FFTCoefs.c
* Purpose : Coefficients used in FFT computation with cR2FFT_q15_q15 function.
*-------------------------------------------------------------
*/

//#define CFFT16
//#define CFFT32
//#define CFFT64
//#define CFFT128
#define CFFT256
//#define CFFT512
//#define CFFT1024



/*****************************************************************************/
#ifdef CFFT16
#pragma ghs section data=".xdata"
short coefOd [8] =
{
 30273, -12539,  12539, -30273, -12539, -30273, -30273, -12539
};
#pragma ghs section data= default
#pragma ghs section data=".ydata"
short coefEv [8] =
{
 32767,      0,  23170, -23170,      0, -32767, -23170, -23170
};
#pragma ghs section data= default
#endif // CFFT16
/*****************************************************************************/
#ifdef CFFT32
#pragma ghs section data=".xdata"
short coefOd [16] =
{
 32137,  -6393,  27245, -18204,  18204, -27245,   6393, -32137,
 -6393, -32137, -18204, -27245, -27245, -18204, -32137,  -6393
};
#pragma ghs section data= default
#pragma ghs section data=".ydata"
short coefEv [16] =
{
 32767,      0,  30273, -12539,  23170, -23170,  12539, -30273,
     0, -32767, -12539, -30273, -23170, -23170, -30273, -12539
};
#pragma ghs section data= default
#endif // CFFT32
/*****************************************************************************/
#ifdef CFFT64
#pragma ghs section data=".xdata"
short coefOd [32] =
{
 32609,  -3212,  31356,  -9512,  28898, -15446,  25329, -20787,
 20787, -25329,  15446, -28898,   9512, -31356,   3212, -32609,
 -3212, -32609,  -9512, -31356, -15446, -28898, -20787, -25329,
-25329, -20787, -28898, -15446, -31356,  -9512, -32609,  -3212
};
#pragma ghs section data= default
#pragma ghs section data=".ydata"
```

```
short coefEv [32] =
{
 32767,      0,  32137,  -6393,  30273, -12539,  27245, -18204,
 23170, -23170,  18204, -27245,  12539, -30273,   6393, -32137,
     0, -32767,  -6393, -32137, -12539, -30273, -18204, -27245,
-23170, -23170, -27245, -18204, -30273, -12539, -32137,  -6393
};
#pragma ghs section data= default
#endif // CFFT64

/*****************************************************************************/
#ifdef CFFT128
#pragma ghs section data=".xdata"
short coefOd [64] =
{
 32728,  -1608,  32412,  -4808,  31785,  -7962,  30852, -11039,
 29621, -14010,  28105, -16846,  26319, -19519,  24279, -22005,
 22005, -24279,  19519, -26319,  16846, -28105,  14010, -29621,
 11039, -30852,   7962, -31785,   4808, -32412,   1608, -32728,
 -1608, -32728,  -4808, -32412,  -7962, -31785, -11039, -30852,
-14010, -29621, -16846, -28105, -19519, -26319, -22005, -24279,
-24279, -22005, -26319, -19519, -28105, -16846, -29621, -14010,
-30852, -11039, -31785,  -7962, -32412,  -4808, -32728,  -1608
};
#pragma ghs section data= default
#pragma ghs section data=".ydata"
short coefEv [64] =
{
 32767,      0,  32609,  -3212,  32137,  -6393,  31356,  -9512,
 30273, -12539,  28898, -15446,  27245, -18204,  25329, -20787,
 23170, -23170,  20787, -25329,  18204, -27245,  15446, -28898,
 12539, -30273,   9512, -31356,   6393, -32137,   3212, -32609,
     0, -32767,  -3212, -32609,  -6393, -32137,  -9512, -31356,
-12539, -30273, -15446, -28898, -18204, -27245, -20787, -25329,
-23170, -23170, -25329, -20787, -27245, -18204, -28898, -15446,
-30273, -12539, -31356,  -9512, -32137,  -6393, -32609,  -3212
};
#pragma ghs section data= default
#endif // CFFT128

/*****************************************************************************/
#ifdef CFFT256
#pragma ghs section data=".xdata"
short coefOd [128] =
{
 32757,   -804,  32678,  -2410,  32521,  -4011,  32285,  -5602,
 31971,  -7179,  31580,  -8739,  31113, -10278,  30571, -11793,
 29956, -13279,  29268, -14732,  28510, -16151,  27683, -17530,
 26790, -18868,  25832, -20159,  24811, -21403,  23731, -22594,
 22594, -23731,  21403, -24811,  20159, -25832,  18868, -26790,
 17530, -27683,  16151, -28510,  14732, -29268,  13279, -29956,
 11793, -30571,  10278, -31113,   8739, -31580,   7179, -31971,
  5602, -32285,   4011, -32521,   2410, -32678,    804, -32757,
  -804, -32757,  -2410, -32678,  -4011, -32521,  -5602, -32285,
 -7179, -31971,  -8739, -31580, -10278, -31113, -11793, -30571,
-13279, -29956, -14732, -29268, -16151, -28510, -17530, -27683,
-18868, -26790, -20159, -25832, -21403, -24811, -22594, -23731,
-23731, -22594, -24811, -21403, -25832, -20159, -26790, -18868,
-27683, -17530, -28510, -16151, -29268, -14732, -29956, -13279,
-30571, -11793, -31113, -10278, -31580,  -8739, -31971,  -7179,
-32285,  -5602, -32521,  -4011, -32678,  -2410, -32757,   -804
};
#pragma ghs section data= default
```

```
#pragma ghs section data=".ydata"
short coefEv [128] =
{
 32767,      0,  32728,  -1608,  32609,  -3212,  32412,  -4808,
 32137,  -6393,  31785,  -7962,  31356,  -9512,  30852, -11039,
 30273, -12539,  29621, -14010,  28898, -15446,  28105, -16846,
 27245, -18204,  26319, -19519,  25329, -20787,  24279, -22005,
 23170, -23170,  22005, -24279,  20787, -25329,  19519, -26319,
 18204, -27245,  16846, -28105,  15446, -28898,  14010, -29621,
 12539, -30273,  11039, -30852,   9512, -31356,   7962, -31785,
  6393, -32137,   4808, -32412,   3212, -32609,   1608, -32728,
     0, -32767,  -1608, -32728,  -3212, -32609,  -4808, -32412,
 -6393, -32137,  -7962, -31785,  -9512, -31356, -11039, -30852,
-12539, -30273, -14010, -29621, -15446, -28898, -16846, -28105,
-18204, -27245, -19519, -26319, -20787, -25329, -22005, -24279,
-23170, -23170, -24279, -22005, -25329, -20787, -26319, -19519,
-27245, -18204, -28105, -16846, -28898, -15446, -29621, -14010,
-30273, -12539, -30852, -11039, -31356,  -9512, -31785,  -7962,
-32137,  -6393, -32412,  -4808, -32609,  -3212, -32728,  -1608
};
#pragma ghs section data= default
#endif // CFFT256
/*****************************************************************************/
/*****************************************************************************/
#ifdef CFFT512
#pragma ghs section data=".xdata"
short coefOd [256] =
{
 32765,   -402,  32745,  -1206,  32705,  -2009,  32646,  -2811,
 32567,  -3612,  32469,  -4410,  32351,  -5205,  32213,  -5998,
 32057,  -6786,  31880,  -7571,  31685,  -8351,  31470,  -9126,
 31237,  -9896,  30985, -10659,  30714, -11417,  30424, -12167,
 30117, -12910,  29791, -13645,  29447, -14372,  29085, -15090,
 28706, -15800,  28310, -16499,  27896, -17189,  27466, -17869,
 27019, -18537,  26556, -19195,  26077, -19841,  25582, -20475,
 25072, -21096,  24547, -21705,  24007, -22301,  23452, -22884,
 22884, -23452,  22301, -24007,  21705, -24547,  21096, -25072,
 20475, -25582,  19841, -26077,  19195, -26556,  18537, -27019,
 17869, -27466,  17189, -27896,  16499, -28310,  15800, -28706,
 15090, -29085,  14372, -29447,  13645, -29791,  12910, -30117,
 12167, -30424,  11417, -30714,  10659, -30985,   9896, -31237,
  9126, -31470,   8351, -31685,   7571, -31880,   6786, -32057,
  5998, -32213,   5205, -32351,   4410, -32469,   3612, -32567,
  2811, -32646,   2009, -32705,   1206, -32745,    402, -32765,
  -402, -32765,  -1206, -32745,  -2009, -32705,  -2811, -32646,
 -3612, -32567,  -4410, -32469,  -5205, -32351,  -5998, -32213,
 -6786, -32057,  -7571, -31880,  -8351, -31685,  -9126, -31470,
 -9896, -31237, -10659, -30985, -11417, -30714, -12167, -30424,
-12910, -30117, -13645, -29791, -14372, -29447, -15090, -29085,
-15800, -28706, -16499, -28310, -17189, -27896, -17869, -27466,
-18537, -27019, -19195, -26556, -19841, -26077, -20475, -25582,
-21096, -25072, -21705, -24547, -22301, -24007, -22884, -23452,
-23452, -22884, -24007, -22301, -24547, -21705, -25072, -21096,
-25582, -20475, -26077, -19841, -26556, -19195, -27019, -18537,
-27466, -17869, -27896, -17189, -28310, -16499, -28706, -15800,
-29085, -15090, -29447, -14372, -29791, -13645, -30117, -12910,
-30424, -12167, -30714, -11417, -30985, -10659, -31237,  -9896,
-31470,  -9126, -31685,  -8351, -31880,  -7571, -32057,  -6786,
-32213,  -5998, -32351,  -5205, -32469,  -4410, -32567,  -3612,
-32646,  -2811, -32705,  -2009, -32745,  -1206, -32765,   -402
};
#pragma ghs section data= default
#pragma ghs section data=".ydata"
```

```
short coefEv [256] =
{
 32767,      0,  32757,   -804,  32728,  -1608,  32678,  -2410,
 32609,  -3212,  32521,  -4011,  32412,  -4808,  32285,  -5602,
 32137,  -6393,  31971,  -7179,  31785,  -7962,  31580,  -8739,
 31356,  -9512,  31113, -10278,  30852, -11039,  30571, -11793,
 30273, -12539,  29956, -13279,  29621, -14010,  29268, -14732,
 28898, -15446,  28510, -16151,  28105, -16846,  27683, -17530,
 27245, -18204,  26790, -18868,  26319, -19519,  25832, -20159,
 25329, -20787,  24811, -21403,  24279, -22005,  23731, -22594,
 23170, -23170,  22594, -23731,  22005, -24279,  21403, -24811,
 20787, -25329,  20159, -25832,  19519, -26319,  18868, -26790,
 18204, -27245,  17530, -27683,  16846, -28105,  16151, -28510,
 15446, -28898,  14732, -29268,  14010, -29621,  13279, -29956,
 12539, -30273,  11793, -30571,  11039, -30852,  10278, -31113,
  9512, -31356,   8739, -31580,   7962, -31785,   7179, -31971,
  6393, -32137,   5602, -32285,   4808, -32412,   4011, -32521,
  3212, -32609,   2410, -32678,   1608, -32728,    804, -32757,
     0, -32767,   -804, -32757,  -1608, -32728,  -2410, -32678,
 -3212, -32609,  -4011, -32521,  -4808, -32412,  -5602, -32285,
 -6393, -32137,  -7179, -31971,  -7962, -31785,  -8739, -31580,
 -9512, -31356, -10278, -31113, -11039, -30852, -11793, -30571,
-12539, -30273, -13279, -29956, -14010, -29621, -14732, -29268,
-15446, -28898, -16151, -28510, -16846, -28105, -17530, -27683,
-18204, -27245, -18868, -26790, -19519, -26319, -20159, -25832,
-20787, -25329, -21403, -24811, -22005, -24279, -22594, -23731,
-23170, -23170, -23731, -22594, -24279, -22005, -24811, -21403,
-25329, -20787, -25832, -20159, -26319, -19519, -26790, -18868,
-27245, -18204, -27683, -17530, -28105, -16846, -28510, -16151,
-28898, -15446, -29268, -14732, -29621, -14010, -29956, -13279,
-30273, -12539, -30571, -11793, -30852, -11039, -31113, -10278,
-31356,  -9512, -31580,  -8739, -31785,  -7962, -31971,  -7179,
-32137,  -6393, -32285,  -5602, -32412,  -4808, -32521,  -4011,
-32609,  -3212, -32678,  -2410, -32728,  -1608, -32757,   -804
};
#pragma ghs section data= default
#endif // CFFT512
/*****************************************************************************/
/*****************************************************************************/
#ifdef CFFT1024
#pragma ghs section data=".xdata"
short coefOd [512] =
{
 32766,   -201,  32761,   -603,  32752,  -1005,  32737,  -1407,
 32717,  -1809,  32692,  -2210,  32663,  -2611,  32628,  -3012,
 32589,  -3412,  32545,  -3811,  32495,  -4210,  32441,  -4609,
 32382,  -5007,  32318,  -5404,  32250,  -5800,  32176,  -6195,
 32098,  -6590,  32014,  -6983,  31926,  -7375,  31833,  -7767,
 31736,  -8157,  31633,  -8545,  31526,  -8933,  31414,  -9319,
 31297,  -9704,  31176, -10087,  31050, -10469,  30919, -10849,
 30783, -11228,  30643, -11605,  30498, -11980,  30349, -12353,
 30195, -12725,  30037, -13094,  29874, -13462,  29706, -13828,
 29534, -14191,  29358, -14553,  29177, -14912,  28992, -15269,
 28803, -15623,  28609, -15976,  28411, -16325,  28208, -16673,
 28001, -17018,  27790, -17360,  27575, -17700,  27356, -18037,
 27133, -18371,  26905, -18703,  26674, -19032,  26438, -19357,
 26198, -19680,  25955, -20000,  25708, -20317,  25456, -20631,
 25201, -20942,  24942, -21250,  24680, -21554,  24413, -21856,
 24143, -22154,  23870, -22448,  23592, -22739,  23311, -23027,
 23027, -23311,  22739, -23592,  22448, -23870,  22154, -24143,
 21856, -24413,  21554, -24680,  21250, -24942,  20942, -25201,
 20631, -25456,  20317, -25708,  20000, -25955,  19680, -26198,
 19357, -26438,  19032, -26674,  18703, -26905,  18371, -27133,
```

```
 18037, -27356,  17700, -27575,  17360, -27790,  17018, -28001,
 16673, -28208,  16325, -28411,  15976, -28609,  15623, -28803,
 15269, -28992,  14912, -29177,  14553, -29358,  14191, -29534,
 13828, -29706,  13462, -29874,  13094, -30037,  12725, -30195,
 12353, -30349,  11980, -30498,  11605, -30643,  11228, -30783,
 10849, -30919,  10469, -31050,  10087, -31176,   9704, -31297,
  9319, -31414,   8933, -31526,   8545, -31633,   8157, -31736,
  7767, -31833,   7375, -31926,   6983, -32014,   6590, -32098,
  6195, -32176,   5800, -32250,   5404, -32318,   5007, -32382,
  4609, -32441,   4210, -32495,   3811, -32545,   3412, -32589,
  3012, -32628,   2611, -32663,   2210, -32692,   1809, -32717,
  1407, -32737,   1005, -32752,    603, -32761,    201, -32766,
  -201, -32766,   -603, -32761,  -1005, -32752,  -1407, -32737,
 -1809, -32717,  -2210, -32692,  -2611, -32663,  -3012, -32628,
 -3412, -32589,  -3811, -32545,  -4210, -32495,  -4609, -32441,
 -5007, -32382,  -5404, -32318,  -5800, -32250,  -6195, -32176,
 -6590, -32098,  -6983, -32014,  -7375, -31926,  -7767, -31833,
 -8157, -31736,  -8545, -31633,  -8933, -31526,  -9319, -31414,
 -9704, -31297, -10087, -31176, -10469, -31050, -10849, -30919,
-11228, -30783, -11605, -30643, -11980, -30498, -12353, -30349,
-12725, -30195, -13094, -30037, -13462, -29874, -13828, -29706,
-14191, -29534, -14553, -29358, -14912, -29177, -15269, -28992,
-15623, -28803, -15976, -28609, -16325, -28411, -16673, -28208,
-17018, -28001, -17360, -27790, -17700, -27575, -18037, -27356,
-18371, -27133, -18703, -26905, -19032, -26674, -19357, -26438,
-19680, -26198, -20000, -25955, -20317, -25708, -20631, -25456,
-20942, -25201, -21250, -24942, -21554, -24680, -21856, -24413,
-22154, -24143, -22448, -23870, -22739, -23592, -23027, -23311,
-23311, -23027, -23592, -22739, -23870, -22448, -24143, -22154,
-24413, -21856, -24680, -21554, -24942, -21250, -25201, -20942,
-25456, -20631, -25708, -20317, -25955, -20000, -26198, -19680,
-26438, -19357, -26674, -19032, -26905, -18703, -27133, -18371,
-27356, -18037, -27575, -17700, -27790, -17360, -28001, -17018,
-28208, -16673, -28411, -16325, -28609, -15976, -28803, -15623,
-28992, -15269, -29177, -14912, -29358, -14553, -29534, -14191,
-29706, -13828, -29874, -13462, -30037, -13094, -30195, -12725,
-30349, -12353, -30498, -11980, -30643, -11605, -30783, -11228,
-30919, -10849, -31050, -10469, -31176, -10087, -31297,  -9704,
-31414,  -9319, -31526,  -8933, -31633,  -8545, -31736,  -8157,
-31833,  -7767, -31926,  -7375, -32014,  -6983, -32098,  -6590,
-32176,  -6195, -32250,  -5800, -32318,  -5404, -32382,  -5007,
-32441,  -4609, -32495,  -4210, -32545,  -3811, -32589,  -3412,
-32628,  -3012, -32663,  -2611, -32692,  -2210, -32717,  -1809,
-32737,  -1407, -32752,  -1005, -32761,   -603, -32766,   -201
};
#pragma ghs section data= default
#pragma ghs section data=".ydata"
short coefEv [512] =
{
 32767,      0,  32765,   -402,  32757,   -804,  32745,  -1206,
 32728,  -1608,  32705,  -2009,  32678,  -2410,  32646,  -2811,
 32609,  -3212,  32567,  -3612,  32521,  -4011,  32469,  -4410,
 32412,  -4808,  32351,  -5205,  32285,  -5602,  32213,  -5998,
 32137,  -6393,  32057,  -6786,  31971,  -7179,  31880,  -7571,
 31785,  -7962,  31685,  -8351,  31580,  -8739,  31470,  -9126,
 31356,  -9512,  31237,  -9896,  31113, -10278,  30985, -10659,
 30852, -11039,  30714, -11417,  30571, -11793,  30424, -12167,
 30273, -12539,  30117, -12910,  29956, -13279,  29791, -13645,
 29621, -14010,  29447, -14372,  29268, -14732,  29085, -15090,
 28898, -15446,  28706, -15800,  28510, -16151,  28310, -16499,
 28105, -16846,  27896, -17189,  27683, -17530,  27466, -17869,
 27245, -18204,  27019, -18537,  26790, -18868,  26556, -19195,
 26319, -19519,  26077, -19841,  25832, -20159,  25582, -20475,
```

```
 25329, -20787,  25072, -21096,  24811, -21403,  24547, -21705,
 24279, -22005,  24007, -22301,  23731, -22594,  23452, -22884,
 23170, -23170,  22884, -23452,  22594, -23731,  22301, -24007,
 22005, -24279,  21705, -24547,  21403, -24811,  21096, -25072,
 20787, -25329,  20475, -25582,  20159, -25832,  19841, -26077,
 19519, -26319,  19195, -26556,  18868, -26790,  18537, -27019,
 18204, -27245,  17869, -27466,  17530, -27683,  17189, -27896,
 16846, -28105,  16499, -28310,  16151, -28510,  15800, -28706,
 15446, -28898,  15090, -29085,  14732, -29268,  14372, -29447,
 14010, -29621,  13645, -29791,  13279, -29956,  12910, -30117,
 12539, -30273,  12167, -30424,  11793, -30571,  11417, -30714,
 11039, -30852,  10659, -30985,  10278, -31113,   9896, -31237,
  9512, -31356,   9126, -31470,   8739, -31580,   8351, -31685,
  7962, -31785,   7571, -31880,   7179, -31971,   6786, -32057,
  6393, -32137,   5998, -32213,   5602, -32285,   5205, -32351,
  4808, -32412,   4410, -32469,   4011, -32521,   3612, -32567,
  3212, -32609,   2811, -32646,   2410, -32678,   2009, -32705,
  1608, -32728,   1206, -32745,    804, -32757,    402, -32765,
     0, -32767,   -402, -32765,   -804, -32757,  -1206, -32745,
 -1608, -32728,  -2009, -32705,  -2410, -32678,  -2811, -32646,
 -3212, -32609,  -3612, -32567,  -4011, -32521,  -4410, -32469,
 -4808, -32412,  -5205, -32351,  -5602, -32285,  -5998, -32213,
 -6393, -32137,  -6786, -32057,  -7179, -31971,  -7571, -31880,
 -7962, -31785,  -8351, -31685,  -8739, -31580,  -9126, -31470,
 -9512, -31356,  -9896, -31237, -10278, -31113, -10659, -30985,
-11039, -30852, -11417, -30714, -11793, -30571, -12167, -30424,
-12539, -30273, -12910, -30117, -13279, -29956, -13645, -29791,
-14010, -29621, -14372, -29447, -14732, -29268, -15090, -29085,
-15446, -28898, -15800, -28706, -16151, -28510, -16499, -28310,
-16846, -28105, -17189, -27896, -17530, -27683, -17869, -27466,
-18204, -27245, -18537, -27019, -18868, -26790, -19195, -26556,
-19519, -26319, -19841, -26077, -20159, -25832, -20475, -25582,
-20787, -25329, -21096, -25072, -21403, -24811, -21705, -24547,
-22005, -24279, -22301, -24007, -22594, -23731, -22884, -23452,
-23170, -23170, -23452, -22884, -23731, -22594, -24007, -22301,
-24279, -22005, -24547, -21705, -24811, -21403, -25072, -21096,
-25329, -20787, -25582, -20475, -25832, -20159, -26077, -19841,
-26319, -19519, -26556, -19195, -26790, -18868, -27019, -18537,
-27245, -18204, -27466, -17869, -27683, -17530, -27896, -17189,
-28105, -16846, -28310, -16499, -28510, -16151, -28706, -15800,
-28898, -15446, -29085, -15090, -29268, -14732, -29447, -14372,
-29621, -14010, -29791, -13645, -29956, -13279, -30117, -12910,
-30273, -12539, -30424, -12167, -30571, -11793, -30714, -11417,
-30852, -11039, -30985, -10659, -31113, -10278, -31237,  -9896,
-31356,  -9512, -31470,  -9126, -31580,  -8739, -31685,  -8351,
-31785,  -7962, -31880,  -7571, -31971,  -7179, -32057,  -6786,
-32137,  -6393, -32213,  -5998, -32285,  -5602, -32351,  -5205,
-32412,  -4808, -32469,  -4410, -32521,  -4011, -32567,  -3612,
-32609,  -3212, -32646,  -2811, -32678,  -2410, -32705,  -2009,
-32728,  -1608, -32745,  -1206, -32757,   -804, -32765,   -402
};
#pragma ghs section data= default
#endif // CFFT1024
/*****************************************************************************/
```

4.4 - cR2FFT3216Coefs.c

```
/*-------------------------------------------------------------
* Project component : ST100 GENLIB
* File Name : cR2FFT3216Coefs.c
* Purpose : Coefficients used in FFT computation with cR2FFT_q31_q15 function.
*-------------------------------------------------------------
*/

#include <st1types.h>

//#define FFT16
//#define FFT32
//#define FFT64
//#define FFT128
#define FFT256
//#define FFT512
//#define FFT1024


#ifdef FFT16
fract16 coefs[16]={
 32767,      0,  30272,  12539,  23169,  23169,  12539,  30272,
     0,  32767, -12539,  30272, -23169,  23169, -30272,  12539
};
#endif
#ifdef FFT32
fract16 coefs[32]={
 32767,      0,  32137,   6392,  30272,  12539,  27244,  18204,
 23169,  23169,  18204,  27244,  12539,  30272,   6392,  32137,
     0,  32767,  -6392,  32137, -12539,  30272, -18204,  27244,
-23169,  23169, -27244,  18204, -30272,  12539, -32137,   6392
};
#endif
#ifdef FFT64
fract16 coefs[64]={
 32767,      0,  32609,   3211,  32137,   6392,  31356,   9511,
 30272,  12539,  28897,  15446,  27244,  18204,  25329,  20787,
 23169,  23169,  20787,  25329,  18204,  27244,  15446,  28897,
 12539,  30272,   9511,  31356,   6392,  32137,   3211,  32609,
     0,  32767,  -3211,  32609,  -6392,  32137,  -9511,  31356,
-12539,  30272, -15446,  28897, -18204,  27244, -20787,  25329,
-23169,  23169, -25329,  20787, -27244,  18204, -28897,  15446,
-30272,  12539, -31356,   9511, -32137,   6392, -32609,   3211
};
#endif
#ifdef FFT128
fract16 coefs[128]={
 32767,      0,  32727,   1607,  32609,   3211,  32412,   4807,
 32137,   6392,  31785,   7961,  31356,   9511,  30851,  11038,
 30272,  12539,  29621,  14009,  28897,  15446,  28105,  16845,
 27244,  18204,  26318,  19519,  25329,  20787,  24278,  22004,
 23169,  23169,  22004,  24278,  20787,  25329,  19519,  26318,
 18204,  27244,  16845,  28105,  15446,  28897,  14009,  29621,
 12539,  30272,  11038,  30851,   9511,  31356,   7961,  31785,
  6392,  32137,   4807,  32412,   3211,  32609,   1607,  32727,
     0,  32767,  -1607,  32727,  -3211,  32609,  -4807,  32412,
 -6392,  32137,  -7961,  31785,  -9511,  31356, -11038,  30851,
-12539,  30272, -14009,  29621, -15446,  28897, -16845,  28105,
-18204,  27244, -19519,  26318, -20787,  25329, -22004,  24278,
-23169,  23169, -24278,  22004, -25329,  20787, -26318,  19519,
-27244,  18204, -28105,  16845, -28897,  15446, -29621,  14009,
-30272,  12539, -30851,  11038, -31356,   9511, -31785,   7961,
```

```
-32137,   6392, -32412,   4807, -32609,   3211, -32727,   1607
};
#endif
#ifdef FFT256
fract16 coefs[256]={
 32767,      0,  32757,    804,  32727,   1607,  32678,   2410,
 32609,   3211,  32520,   4011,  32412,   4807,  32284,   5601,
 32137,   6392,  31970,   7179,  31785,   7961,  31580,   8739,
 31356,   9511,  31113,  10278,  30851,  11038,  30571,  11792,
 30272,  12539,  29955,  13278,  29621,  14009,  29268,  14732,
 28897,  15446,  28510,  16150,  28105,  16845,  27683,  17530,
 27244,  18204,  26789,  18867,  26318,  19519,  25831,  20159,
 25329,  20787,  24811,  21402,  24278,  22004,  23731,  22594,
 23169,  23169,  22594,  23731,  22004,  24278,  21402,  24811,
 20787,  25329,  20159,  25831,  19519,  26318,  18867,  26789,
 18204,  27244,  17530,  27683,  16845,  28105,  16150,  28510,
 15446,  28897,  14732,  29268,  14009,  29621,  13278,  29955,
 12539,  30272,  11792,  30571,  11038,  30851,  10278,  31113,
  9511,  31356,   8739,  31580,   7961,  31785,   7179,  31970,
  6392,  32137,   5601,  32284,   4807,  32412,   4011,  32520,
  3211,  32609,   2410,  32678,   1607,  32727,    804,  32757,
     0,  32767,   -804,  32757,  -1607,  32727,  -2410,  32678,
 -3211,  32609,  -4011,  32520,  -4807,  32412,  -5601,  32284,
 -6392,  32137,  -7179,  31970,  -7961,  31785,  -8739,  31580,
 -9511,  31356, -10278,  31113, -11038,  30851, -11792,  30571,
-12539,  30272, -13278,  29955, -14009,  29621, -14732,  29268,
-15446,  28897, -16150,  28510, -16845,  28105, -17530,  27683,
-18204,  27244, -18867,  26789, -19519,  26318, -20159,  25831,
-20787,  25329, -21402,  24811, -22004,  24278, -22594,  23731,
-23169,  23169, -23731,  22594, -24278,  22004, -24811,  21402,
-25329,  20787, -25831,  20159, -26318,  19519, -26789,  18867,
-27244,  18204, -27683,  17530, -28105,  16845, -28510,  16150,
-28897,  15446, -29268,  14732, -29621,  14009, -29955,  13278,
-30272,  12539, -30571,  11792, -30851,  11038, -31113,  10278,
-31356,   9511, -31580,   8739, -31785,   7961, -31970,   7179,
-32137,   6392, -32284,   5601, -32412,   4807, -32520,   4011,
-32609,   3211, -32678,   2410, -32727,   1607, -32757,    804
};
#endif
#ifdef FFT512
fract16 coefs[512]={
 32767,      0,  32764,    402,  32757,    804,  32744,   1206,
 32727,   1607,  32705,   2009,  32678,   2410,  32646,   2811,
 32609,   3211,  32567,   3611,  32520,   4011,  32468,   4409,
 32412,   4807,  32350,   5205,  32284,   5601,  32213,   5997,
 32137,   6392,  32056,   6786,  31970,   7179,  31880,   7571,
 31785,   7961,  31684,   8351,  31580,   8739,  31470,   9126,
 31356,   9511,  31236,   9895,  31113,  10278,  30984,  10659,
 30851,  11038,  30713,  11416,  30571,  11792,  30424,  12166,
 30272,  12539,  30116,  12909,  29955,  13278,  29790,  13645,
 29621,  14009,  29446,  14372,  29268,  14732,  29085,  15090,
 28897,  15446,  28706,  15799,  28510,  16150,  28309,  16499,
 28105,  16845,  27896,  17189,  27683,  17530,  27466,  17868,
 27244,  18204,  27019,  18537,  26789,  18867,  26556,  19194,
 26318,  19519,  26077,  19840,  25831,  20159,  25582,  20474,
 25329,  20787,  25072,  21096,  24811,  21402,  24546,  21705,
 24278,  22004,  24006,  22301,  23731,  22594,  23452,  22883,
 23169,  23169,  22883,  23452,  22594,  23731,  22301,  24006,
 22004,  24278,  21705,  24546,  21402,  24811,  21096,  25072,
 20787,  25329,  20474,  25582,  20159,  25831,  19840,  26077,
 19519,  26318,  19194,  26556,  18867,  26789,  18537,  27019,
 18204,  27244,  17868,  27466,  17530,  27683,  17189,  27896,
 16845,  28105,  16499,  28309,  16150,  28510,  15799,  28706,
```

```
 15446,  28897,  15090,  29085,  14732,  29268,  14372,  29446,
 14009,  29621,  13645,  29790,  13278,  29955,  12909,  30116,
 12539,  30272,  12166,  30424,  11792,  30571,  11416,  30713,
 11038,  30851,  10659,  30984,  10278,  31113,   9895,  31236,
  9511,  31356,   9126,  31470,   8739,  31580,   8351,  31684,
  7961,  31785,   7571,  31880,   7179,  31970,   6786,  32056,
  6392,  32137,   5997,  32213,   5601,  32284,   5205,  32350,
  4807,  32412,   4409,  32468,   4011,  32520,   3611,  32567,
  3211,  32609,   2811,  32646,   2410,  32678,   2009,  32705,
  1607,  32727,   1206,  32744,    804,  32757,    402,  32764,
     0,  32767,   -402,  32764,   -804,  32757,  -1206,  32744,
 -1607,  32727,  -2009,  32705,  -2410,  32678,  -2811,  32646,
 -3211,  32609,  -3611,  32567,  -4011,  32520,  -4409,  32468,
 -4807,  32412,  -5205,  32350,  -5601,  32284,  -5997,  32213,
 -6392,  32137,  -6786,  32056,  -7179,  31970,  -7571,  31880,
 -7961,  31785,  -8351,  31684,  -8739,  31580,  -9126,  31470,
 -9511,  31356,  -9895,  31236, -10278,  31113, -10659,  30984,
-11038,  30851, -11416,  30713, -11792,  30571, -12166,  30424,
-12539,  30272, -12909,  30116, -13278,  29955, -13645,  29790,
-14009,  29621, -14372,  29446, -14732,  29268, -15090,  29085,
-15446,  28897, -15799,  28706, -16150,  28510, -16499,  28309,
-16845,  28105, -17189,  27896, -17530,  27683, -17868,  27466,
-18204,  27244, -18537,  27019, -18867,  26789, -19194,  26556,
-19519,  26318, -19840,  26077, -20159,  25831, -20474,  25582,
-20787,  25329, -21096,  25072, -21402,  24811, -21705,  24546,
-22004,  24278, -22301,  24006, -22594,  23731, -22883,  23452,
-23169,  23169, -23452,  22883, -23731,  22594, -24006,  22301,
-24278,  22004, -24546,  21705, -24811,  21402, -25072,  21096,
-25329,  20787, -25582,  20474, -25831,  20159, -26077,  19840,
-26318,  19519, -26556,  19194, -26789,  18867, -27019,  18537,
-27244,  18204, -27466,  17868, -27683,  17530, -27896,  17189,
-28105,  16845, -28309,  16499, -28510,  16150, -28706,  15799,
-28897,  15446, -29085,  15090, -29268,  14732, -29446,  14372,
-29621,  14009, -29790,  13645, -29955,  13278, -30116,  12909,
-30272,  12539, -30424,  12166, -30571,  11792, -30713,  11416,
-30851,  11038, -30984,  10659, -31113,  10278, -31236,   9895,
-31356,   9511, -31470,   9126, -31580,   8739, -31684,   8351,
-31785,   7961, -31880,   7571, -31970,   7179, -32056,   6786,
-32137,   6392, -32213,   5997, -32284,   5601, -32350,   5205,
-32412,   4807, -32468,   4409, -32520,   4011, -32567,   3611,
-32609,   3211, -32646,   2811, -32678,   2410, -32705,   2009,
-32727,   1607, -32744,   1206, -32757,    804, -32764,    402
};
#endif
#ifdef FFT1024
fract16 coefs[1024]={
 32767,      0,  32766,    201,  32764,    402,  32761,    603,
 32757,    804,  32751,   1005,  32744,   1206,  32736,   1406,
 32727,   1607,  32717,   1808,  32705,   2009,  32692,   2209,
 32678,   2410,  32662,   2610,  32646,   2811,  32628,   3011,
 32609,   3211,  32588,   3411,  32567,   3611,  32544,   3811,
 32520,   4011,  32495,   4210,  32468,   4409,  32441,   4608,
 32412,   4807,  32382,   5006,  32350,   5205,  32318,   5403,
 32284,   5601,  32249,   5799,  32213,   5997,  32176,   6195,
 32137,   6392,  32097,   6589,  32056,   6786,  32014,   6982,
 31970,   7179,  31926,   7375,  31880,   7571,  31833,   7766,
 31785,   7961,  31735,   8156,  31684,   8351,  31633,   8545,
 31580,   8739,  31525,   8932,  31470,   9126,  31413,   9319,
 31356,   9511,  31297,   9703,  31236,   9895,  31175,  10087,
 31113,  10278,  31049,  10469,  30984,  10659,  30918,  10849,
 30851,  11038,  30783,  11227,  30713,  11416,  30643,  11604,
 30571,  11792,  30498,  11980,  30424,  12166,  30349,  12353,
 30272,  12539,  30195,  12724,  30116,  12909,  30036,  13094,
```

```
 29955,  13278,  29873,  13462,  29790,  13645,  29706,  13827,
 29621,  14009,  29534,  14191,  29446,  14372,  29358,  14552,
 29268,  14732,  29177,  14911,  29085,  15090,  28992,  15268,
 28897,  15446,  28802,  15623,  28706,  15799,  28608,  15975,
 28510,  16150,  28410,  16325,  28309,  16499,  28208,  16672,
 28105,  16845,  28001,  17017,  27896,  17189,  27790,  17360,
 27683,  17530,  27575,  17699,  27466,  17868,  27355,  18036,
 27244,  18204,  27132,  18371,  27019,  18537,  26905,  18702,
 26789,  18867,  26673,  19031,  26556,  19194,  26437,  19357,
 26318,  19519,  26198,  19680,  26077,  19840,  25954,  20000,
 25831,  20159,  25707,  20317,  25582,  20474,  25456,  20631,
 25329,  20787,  25201,  20942,  25072,  21096,  24942,  21249,
 24811,  21402,  24679,  21554,  24546,  21705,  24413,  21855,
 24278,  22004,  24143,  22153,  24006,  22301,  23869,  22448,
 23731,  22594,  23592,  22739,  23452,  22883,  23311,  23027,
 23169,  23169,  23027,  23311,  22883,  23452,  22739,  23592,
 22594,  23731,  22448,  23869,  22301,  24006,  22153,  24143,
 22004,  24278,  21855,  24413,  21705,  24546,  21554,  24679,
 21402,  24811,  21249,  24942,  21096,  25072,  20942,  25201,
 20787,  25329,  20631,  25456,  20474,  25582,  20317,  25707,
 20159,  25831,  20000,  25954,  19840,  26077,  19680,  26198,
 19519,  26318,  19357,  26437,  19194,  26556,  19031,  26673,
 18867,  26789,  18702,  26905,  18537,  27019,  18371,  27132,
 18204,  27244,  18036,  27355,  17868,  27466,  17699,  27575,
 17530,  27683,  17360,  27790,  17189,  27896,  17017,  28001,
 16845,  28105,  16672,  28208,  16499,  28309,  16325,  28410,
 16150,  28510,  15975,  28608,  15799,  28706,  15623,  28802,
 15446,  28897,  15268,  28992,  15090,  29085,  14911,  29177,
 14732,  29268,  14552,  29358,  14372,  29446,  14191,  29534,
 14009,  29621,  13827,  29706,  13645,  29790,  13462,  29873,
 13278,  29955,  13094,  30036,  12909,  30116,  12724,  30195,
 12539,  30272,  12353,  30349,  12166,  30424,  11980,  30498,
 11792,  30571,  11604,  30643,  11416,  30713,  11227,  30783,
 11038,  30851,  10849,  30918,  10659,  30984,  10469,  31049,
 10278,  31113,  10087,  31175,   9895,  31236,   9703,  31297,
  9511,  31356,   9319,  31413,   9126,  31470,   8932,  31525,
  8739,  31580,   8545,  31633,   8351,  31684,   8156,  31735,
  7961,  31785,   7766,  31833,   7571,  31880,   7375,  31926,
  7179,  31970,   6982,  32014,   6786,  32056,   6589,  32097,
  6392,  32137,   6195,  32176,   5997,  32213,   5799,  32249,
  5601,  32284,   5403,  32318,   5205,  32350,   5006,  32382,
  4807,  32412,   4608,  32441,   4409,  32468,   4210,  32495,
  4011,  32520,   3811,  32544,   3611,  32567,   3411,  32588,
  3211,  32609,   3011,  32628,   2811,  32646,   2610,  32662,
  2410,  32678,   2209,  32692,   2009,  32705,   1808,  32717,
  1607,  32727,   1406,  32736,   1206,  32744,   1005,  32751,
   804,  32757,    603,  32761,    402,  32764,    201,  32766,
     0,  32767,   -201,  32766,   -402,  32764,   -603,  32761,
  -804,  32757,  -1005,  32751,  -1206,  32744,  -1406,  32736,
 -1607,  32727,  -1808,  32717,  -2009,  32705,  -2209,  32692,
 -2410,  32678,  -2610,  32662,  -2811,  32646,  -3011,  32628,
 -3211,  32609,  -3411,  32588,  -3611,  32567,  -3811,  32544,
 -4011,  32520,  -4210,  32495,  -4409,  32468,  -4608,  32441,
 -4807,  32412,  -5006,  32382,  -5205,  32350,  -5403,  32318,
 -5601,  32284,  -5799,  32249,  -5997,  32213,  -6195,  32176,
 -6392,  32137,  -6589,  32097,  -6786,  32056,  -6982,  32014,
 -7179,  31970,  -7375,  31926,  -7571,  31880,  -7766,  31833,
 -7961,  31785,  -8156,  31735,  -8351,  31684,  -8545,  31633,
 -8739,  31580,  -8932,  31525,  -9126,  31470,  -9319,  31413,
 -9511,  31356,  -9703,  31297,  -9895,  31236, -10087,  31175,
-10278,  31113, -10469,  31049, -10659,  30984, -10849,  30918,
-11038,  30851, -11227,  30783, -11416,  30713, -11604,  30643,
-11792,  30571, -11980,  30498, -12166,  30424, -12353,  30349,
```

```
-12539,  30272, -12724,  30195, -12909,  30116, -13094,  30036,
-13278,  29955, -13462,  29873, -13645,  29790, -13827,  29706,
-14009,  29621, -14191,  29534, -14372,  29446, -14552,  29358,
-14732,  29268, -14911,  29177, -15090,  29085, -15268,  28992,
-15446,  28897, -15623,  28802, -15799,  28706, -15975,  28608,
-16150,  28510, -16325,  28410, -16499,  28309, -16672,  28208,
-16845,  28105, -17017,  28001, -17189,  27896, -17360,  27790,
-17530,  27683, -17699,  27575, -17868,  27466, -18036,  27355,
-18204,  27244, -18371,  27132, -18537,  27019, -18702,  26905,
-18867,  26789, -19031,  26673, -19194,  26556, -19357,  26437,
-19519,  26318, -19680,  26198, -19840,  26077, -20000,  25954,
-20159,  25831, -20317,  25707, -20474,  25582, -20631,  25456,
-20787,  25329, -20942,  25201, -21096,  25072, -21249,  24942,
-21402,  24811, -21554,  24679, -21705,  24546, -21855,  24413,
-22004,  24278, -22153,  24143, -22301,  24006, -22448,  23869,
-22594,  23731, -22739,  23592, -22883,  23452, -23027,  23311,
-23169,  23169, -23311,  23027, -23452,  22883, -23592,  22739,
-23731,  22594, -23869,  22448, -24006,  22301, -24143,  22153,
-24278,  22004, -24413,  21855, -24546,  21705, -24679,  21554,
-24811,  21402, -24942,  21249, -25072,  21096, -25201,  20942,
-25329,  20787, -25456,  20631, -25582,  20474, -25707,  20317,
-25831,  20159, -25954,  20000, -26077,  19840, -26198,  19680,
-26318,  19519, -26437,  19357, -26556,  19194, -26673,  19031,
-26789,  18867, -26905,  18702, -27019,  18537, -27132,  18371,
-27244,  18204, -27355,  18036, -27466,  17868, -27575,  17699,
-27683,  17530, -27790,  17360, -27896,  17189, -28001,  17017,
-28105,  16845, -28208,  16672, -28309,  16499, -28410,  16325,
-28510,  16150, -28608,  15975, -28706,  15799, -28802,  15623,
-28897,  15446, -28992,  15268, -29085,  15090, -29177,  14911,
-29268,  14732, -29358,  14552, -29446,  14372, -29534,  14191,
-29621,  14009, -29706,  13827, -29790,  13645, -29873,  13462,
-29955,  13278, -30036,  13094, -30116,  12909, -30195,  12724,
-30272,  12539, -30349,  12353, -30424,  12166, -30498,  11980,
-30571,  11792, -30643,  11604, -30713,  11416, -30783,  11227,
-30851,  11038, -30918,  10849, -30984,  10659, -31049,  10469,
-31113,  10278, -31175,  10087, -31236,   9895, -31297,   9703,
-31356,   9511, -31413,   9319, -31470,   9126, -31525,   8932,
-31580,   8739, -31633,   8545, -31684,   8351, -31735,   8156,
-31785,   7961, -31833,   7766, -31880,   7571, -31926,   7375,
-31970,   7179, -32014,   6982, -32056,   6786, -32097,   6589,
-32137,   6392, -32176,   6195, -32213,   5997, -32249,   5799,
-32284,   5601, -32318,   5403, -32350,   5205, -32382,   5006,
-32412,   4807, -32441,   4608, -32468,   4409, -32495,   4210,
-32520,   4011, -32544,   3811, -32567,   3611, -32588,   3411,
-32609,   3211, -32628,   3011, -32646,   2811, -32662,   2610,
-32678,   2410, -32692,   2209, -32705,   2009, -32717,   1808,
-32727,   1607, -32736,   1406, -32744,   1206, -32751,   1005,
-32757,    804, -32761,    603, -32764,    402, -32766,    201
};
#endif
```

4.5 - aCorrScal_q15

```
/*-------------------------------------------------------------
* FUNCTION : aCorrScal_q15
* Project component : ST100 GENLIB
* File Name : aCorrScal.s
* Purpose : Auto correlation of a vector.
*-------------------------------------------------------------
*/

 .text
 .entry32
 .globl aCorrScal_q15

/*-------------------------------------------------------------
* Function name: void aCorrScal_q15(fract16 *outData,fract16 *inData,
*                                   int16 nbOut, int16 nbIn, int16 scale)
*
* Purpose: auto correlation
*
* In:
*      - outData address of output data
*      - inData address of input data
*      - nbOut number of output data
*      - nbIn number of input data
*      - scale Partial products are scaled by scale bits
* Out:
*      - outData output data
* In/Out:
*      -
* Read global variables
*      -
* Modified global variables
*      -
* Returns:
*      -
*-------------------------------------------------------------
*----- Input Parameters-----

* p1 <- InputData Table:Input
* p0 <- AutoCorrData Table:Output
* R1 <- Data Size: size of input data: short
* R0 <- NumberOfFlags: size of output data: short
* R2 <- Scale: scalinf factor: short
* No return value
*
*
* -----Local Variables-------
*
* R1 <- LastIndex. Data Size does not need to be kept. We use R1 to perform LastIndex =
LastIndex--
* P2 <- lag
* R13 <- Accumulator 1
* R12 <- Accumulator 2
* R14 <- to load InputData[i]
* R15 <- to load InputData[i+lag]
* R0 <- Used to initialize the loop counter of the inner loop
*       once the outer loop counter has been initialized
*
*
*---------Caution--------------
* To improve performance, we use two accumulators in the inner loop
```

```
* so instead of loading half-word data, we load 2 half-word data
* InputData[i]& InputData[i+1], InputData[i+lag]&InputData[i+1+lag]
* Number of iterations of the inner loop (LastIndex) is thus divided by two.
* Before doing this, we must be sure of the parity of LastIndex
* When LastIndex is odd (g0 is set), a first step is done outside the inner loop
* LastIndex is decremented by one before being divided by two.
* When LastIndex is even, both accumulators are merely reset.
*
*
*/

aCorrScal_q15:
.align 8
push LR1,r4-r5
copyc LC0,R0            // Outer Loop Counter initialisation

tbpos g0,R1,0           // Checking parity of DataSize g0 is set if DataSize is odd
                        // For the first iteration of the outer loop, LastIndex = DataSize
                        // setls0 outer_begin

add R0,R1,0             // Preparing the initialization of LC1 for the first iteration of
                        // the outer loop
setle0 (outer_end-16)

g0 ? sub R0,R1,1        // If DataSize is odd, LastIndex=DataSize-1
setls1 inner_begin

shr R0,R0,1             // Dividing by two LastIndex,first number of iterations of the
                        // innerloop
setle1 (inner_end-16)

copya p14,R1            // Initializes the pointer enabling the
                        // re initialization of InputData before entering the inner loop
copyc LC1, r0           // Initializes the inner loop counter for the first iteration of
                        // the outer loop


.presliw
sliwmd

make r4,0               // resetting the accumulator 1
make r5,0
makea p2,0              // initialization of lag to load InputData[i+lag]
addba p3,p1,0



//.align 16
outer_begin:


g0 ? ldh r14,@(P3 ?+2 )
g0 ? ldh r15,@(p1 ?+ 2)
g0 ? mpfll r13,r14,r15
     nop

g0 ? shr    r5,r13,r2
     nop
     nop
     nop

         //.align 16
         inner_begin:
```

```
                ldw r15,@(P3!+4)          // Load InputData[i+lag]
                ldw r14,@(p1!+4)          // Load InputData[i] and increment p1
                mpfll  r13,r15,r14
                mpfhh  r12, r15,r14

                shr    r13,r13,r2         // shift Right by Scale Value
                shr    r12,r12,r2
                nop
                nop

                add    r4,r4,r13
                add    r5,r5,r12
                nop
                nop

    //    .align 8
        inner_end:

                        // New Initialization of the loop counter for the inner loop
                        // DataSizeUpdate-1

sub R1, R1,1            // Performs LastIndex= DataSize - lag = DataSize --
add r4,r4,r5            // Add both intermediate accumulators to get AutoCorrData[lag]
nop
sdf @(p0!+2),r4         // Storing AutoCorrData[lag], high part of r13

sub R0,R1, 1            // Assuming new value of R1 is odd
nop
subha p1,p1,p14         // Re initializes p1 pointer at the beginning of InputData
nop

shr R0,R0,1             // Getting the new value for LC1 in case R1 is odd
nop
nop
nop

nop
g0 ? shr R0,R1,1        // Crashing the previous value of r1 if ro turns to be even
addba P2,P2,1           // Computes the new value of lag in the AU
nop

nandg g0,g0,g0          // Now parity of R1 is the invert of its previous value so we
                        // complement g0
nop
copyc LC1, R0           // Re initializing the inner loop counter
subba p14,p14,1         // Performing DataSize - lag for re initializing p1

make r5,0
make r4,0               // Resetting the value of the accumulator 1
nop
addha p3,p1,p2


//.align 8
outer_end:

     nop
     nop
     nop
     gp32md

poprts LR1,r4-r5
```

```
.size aCorrScal_q15,.-aCorrScal_q15
.type aCorrScal_q15,@function
.globl __callee.aCorrScal_q15.v.ppiii
```

4.6 - bitRev_16

```
/*-------------------------------------------------------------
* FUNCTION : bitRev_16
* Project component : ST100 GENLIB
* File Name : bitRev_16.s
* Purpose : In place  bit-reversing of a vector of nvec 16-bit words.
*-------------------------------------------------------------
*/

         .text
         .entry32
         .globl           bitRev_16

/*-------------------------------------------------------------
* Function name: bitRev_16 (int16 *vec, int16 nvec)
*
* Purpose: In place bit-reversing elements in a vector
* In:
*      - vec Address of the vector
*      - nvec Size of the vector (multiple of 4)
* Out:
*      - vec Address of the bit reversed elements
* In/Out:
*      -
* Read global variables
*      -
* Modified global variables
*      -
* Returns:
*      -
*-------------------------------------------------------------
*/

bitRev_16:
         push           P4-P5

         shruw          r1, r0, 2

         copyc          lc0, r1
         copya          p1, r0

         shra1          p14, p1
         addba          p1, p1, p1

         setuls0        _bitrev_ls0
         bitra          p1, p1

         setle0         _bitrev_le0 - 16
         addba          p2, p0, 0

         makea          p4, 0
         addba          p5, p14, 1
         .align 16
         .presliw
         sliwmd
_bitrev_ls0:
         ldhsw          r0, @p0
         movehl         r1, r0
         movell         r2, r0
         bitra          p3, p4
```

```
        nop
        nop
        nop
        addba           p3, p2, p3

        nop
        nop
        nop
        gta             g0, p3, p0

        ldp             r3, @(p3 + p14)
        movelh          r1, r3
        movehh          r2, r3
        addwa           p4, p4, p1

g0?     ldhh            r2, @(p0 + p5)
g0?     ldlh            r1, @p3
        nop
        nop

        nop
        nop
g0?     sdf             @p3, r0
g0?     sdf             @(p0 + p5), r3

        nop
        nop
        sdp             @(p3 + p14), r2
        sdw             @(p0 !+ 4), r1

_bitrev_le0:

        nop
        nop
        nop
        gp32md

        poprts P4-P5

        .type           bitRev_16,@function
        .size           bitRev_16,.-bitRev_16

        .globl          __callee.bitRev_16.v.pi
```

4.7 - bitRev_32

```
/*-------------------------------------------------------------
* FUNCTION : bitRev_32
* Project component : ST100 GENLIB
* File Name : bitRev_32.s
* Purpose : In place bit-reversing of a vector of nvec 32-bit words.
*-------------------------------------------------------------
*/
        .text
        .entry32
        .globl          bitRev_32

/*-------------------------------------------------------------
* Function name : bitRev_32 (int32 *vec, int16 nvec)
*
* Purpose : performs in place bit-reversal of a vector of 32b elements
* In:
*      - vec Address of the vector
*      - nvec Size of the vector (multiple of 4)
* Out :
*      - vec Adress of the bit- reversed elements
* In/Out:
*      -
* Read global variables
*      -
* Modified global variables
*      -
* Returns:
*      -
*-------------------------------------------------------------
*/

bitRev_32:
        push P4-P5
        shrw    R1, R0, 2
        copyc   LC0, R1
        copya   P1, R0

        shra1   P1, P1
        bitra   P14, P1

        setuls0 __bit_reverse_st
        setle0  __bit_reverse_en - 4
        makea   P3, 0
        makea   P12, 0
        makea   P2, 0
        addwa   P1, P0, P1
        addba   P1, P1, 4
        addba   P4, P0, 4
        .align 16

__bit_reverse_st:

        bitra   P3, P12
        ldw     R0, @(P0+P3)

        ldw     R1, @(P0+P2)
        gta     G0, P3, P2

        ldw     R2, @(P1+P3)
        ldw     R3, @(P1+P2)
```

```
G0?      sdw    @(P0+P3), R1
G0?      sdw    @(P0+P2), R0

G0?      sdw    @(P1+P3), R3
         addba  P12, P12, P14

G0?      sdw    @(P1+P2), R2
         bitra  P3, P12

         addba  P12, P12, P14
         ldw    R0, @(P0+P3)

         ldw    R1, @(P4+P2)
         sdw    @(P4+P2), R0

         sdw    @(P0+P3), R1
         addba  P2, P2, 2
         .align 8
__bit_reverse_en:

         poprts P4-P5

         .type  bitRev_32,@function
         .size  bitRev_32,.-bitRev_32

         .globl __callee.bitrev32.v.pi
```

4.8 - cFIR_q15

```
/*-------------------------------------------------------------
* FUNCTION : cFIR_q15
* Project component : ST100 GENLIB
* File Name : cFIR_q15.s
* Purpose : Complex FIR filter (direct form) of a 16-bit fractional vector.
*-------------------------------------------------------------
*/
        .entry32
        .globl  cFIR_q15

/*-------------------------------------------------------------
* Function name : void cFIR_q15(fract16 *out, fract16 *in, fract16 *coefs,
*                    fract16 **delay, int16 nIn, int16 nCoefs, int16 scale);
*
* Purpose: Complex FIR filter
* In:
*      - in address of the input buffer. mem(2n) holds real part of X(n),
*        mem(2n+1) holds imaginary part of X(n)
*      - coefs address of the coefficients
*      - delay pointer on the address of a delay line (size=64 bytes)
*      - nIn number of sample in input buffer
*      - nCoefs number of coefficients (maxi=16)
*      - scale scale bit for each output
* Out:
*      - out address of output buffer
* In/Out:
*      -
* Read global variables
*      -
* Modified global variables
*      -
* Returns:
*      -
*-------------------------------------------------------------
*/
cFIR_q15:
         push        LR1,P4-P5

         SUB         R3,R1,2
         COPYC       LC1, R3                    // LCR1 = (nCoefs - 2)

         SHR         R0,R0,1
         COPYC       LC0, R0                    // LCR0 = nInputs/2

         SETULS0     _block_loop_start          // LSTART0 _block_loop_start
         SETLE0      _block_loop_end - 16       // LVLIWE0 _block_loop_end

         SETULS1     _filter_loop_start         // LSTART1 _filter_loop_start
         SETLE1      _filter_loop_end - 16      // LVLIWE1 _filter_loop_end

         law         P12,@(P15+24)
         law         P3,@(P12+0)

         SUB         R3,R1,1
         COPYA       P4,R3

         ADD         R3,R1,R1
         COPYA       P5,R3

         .presliw
```

```
        SLIWmd                                  // sliwmd


_block_loop_start:

        LDW         R3, @(P2 !+ 4)              // R3 s= [P2 !+ 4]
        LDW         R13, @(P1 !+ 4)             // R13 s= [P1 !+ 4]
        MPFCLL      R0, R13, R3                 // R0 cf= R13L * R3L
        MPFCLH      R1, R13, R3                 // R1 cf= R13L * R3H

        MSFCHH      R0, R0, R13, R3             // R0 cf= R0 - R13H * R3H
        MAFCHL      R1, R1, R13, R3             // R1 cf= R1 + R13H * R3L
        NOP                                     // nop
        SDW         %64 @(P3 !+ 4), R13         // [P3 !+ 4 %64] = R13

_filter_loop_start:

        LDW         R3, @(P2 !+ 4)              // R3 s= [P2 !+ 4]
        LDP         %64 R12, @(P3 !+ 4)         // R12 shp= [P3 !+ 4 %64]
        MAFCLL      R0, R0, R12, R3             // R0 cf= R0 + R12L * R3L
        MAFCLH      R1, R1, R12, R3             // R1 cf= R1 + R12L * R3H

        NOP                                     // nop
        NOP                                     // nop
        MSFCHH      R0, R0, R12, R3             // R0 cf= R0 - R12H * R3H
        MAFCHL      R1, R1, R12, R3             // cf= R1 + R12H * R3L

_filter_loop_end:

        LDW         R3, @(P2 !- P4)             // R3 s= [P2 !- P4]
        LDP         %64 R13, @(P3 !- P5)        // R13 shp= [P3 !- P5 %64]
        MAFCLL      R0, R0, R13, R3             // R0 cf= R0 + R13L * R3L
        MAFCLH      R1, R1, R13, R3             // R1 cf= R1 + R13L * R3H

        NOP                                     // nop
        NOP                                     // nop
        MSFCHH      R0, R0, R13, R3             // R0 cf= R0 - R13H * R3H
        MAFCHL      R1, R1, R13, R3             // R1 cf= R1 + R13H * R3L

        SHR         R0, R0, R2                  // R0 = R0 >> SCALE
        SHR         R1, R1, R2                  // R1 = R1 >> SCALE
        SDF         @(P0 + 2), R1               // [P0 + 2] h= R1H
        SDF         @(P0 !+ 4), R0              // [P0 !+ 4] h= R0H

_block_loop_end:

        NOP                                     // nop
        NOP                                     // nop
        saw         @(P12+0),P3                 // nop
        GP32md                                  // gP32md

        poprts      LR1,P4-P5

        .type       cFIR_q15, @function
        .size       cFIR_q15, .-cFIR_q15
        .globl      __callee.cFIR_q15.v.ppppiii
```

4.9 - conv_q15

```
/*-------------------------------------------------------------
* FUNCTION : conv_q15
* Project component : ST100 GENLIB
* File Name : conv_q15.s
* Purpose : Full length convolution between vector x and h.
*-------------------------------------------------------------
*/

        .text
        .globl          conv_q15
        .entry32

/*-------------------------------------------------------------
* Function name: void conv_q15(fract16 *y, fract16 *x,
*                        fract16 *h, fract16 nx, fract16 nh)
*
* Purpose:Full length convolution between vector x and h.
* In:
*      - x: address of input vector 1
*      - h: address of input vector 2
*      - no: number of element in vector x
*      - nh: number of element in vector y
* Out:
*      - y: address of output vector (size nx+nh-1)
* In/Out :
*      -
* Read global variables
*      -
* Modified global variables
*      -
* Returns:
*      -
*-------------------------------------------------------------
*/

conv_q15:
        copyc   lc1,r1                          // nh
        sub     r1,r1,1                         // nh - 1
        add     r2,r0,r1                        // nx + nh - 1
        copyc   lc0,r2                          // nx + nh - 1
        setuls0 L_outerLoopSt
        setle0  L_outerLoopEn - 16
        setuls1 L_innerLoopSt
        setle1  L_innerLoopEn - 16

        copya   p3,r1
        addha   p12,p2,p3                       // hes = h + nh - 1
        make    r2,0

.presliw
        sliwmd
//.align 16
L_outerLoopSt:
        make    r3,0                            // res = 0
        sub     r4,r2,r1                        // save k
        addha   p2,p12,0                        // he = hes
        subha   p14,p1,p3                       // xd = x - nh + 1
//.align 16
L_innerLoopSt:
        gew     g0,r4,0                         // (j-k)>=0 ?
```

```
        ltw     g1,r4,r0                        // (j-k)<nx
        nop
        nop

        andg    g0,g0,g1
        add     r4,r4,1                         // k--
        nop
        nop

        ldh     r12,@(p2!-2)
        ldh     r13,@(p14!+2)
g0?     mafcll  r3,r3,r12,r13                   // res+= (*he--) * (*xd++)
        nop
//.align 8
L_innerLoopEn:
        nop
        add     r2,r2,1                         // j++
        addba   p1,p1,2                         // x++
        sdf     @(p0!+2),r3                     // *y++=res
//.align 8
L_outerLoopEn:
        nop
        nop
        nop
        gp32md

        poprts


        .type   conv_q15,@function
        .size   conv_q15,.-conv_q15
        .globl  __callee.conv_q15.v.pppii
```

4.10 - conv2_q15

```
/*-------------------------------------------------------------
* FUNCTION : conv2_q15
* Project component : ST100 GENLIB
* File Name : conv2_q15.s
* Purpose : Convolution between vector x and h.
*-------------------------------------------------------------
*/

        .text
        .globl          conv2_q15
        .entry32

/*-------------------------------------------------------------
* Function name: void conv2_q15(fract16 *y, fract16 *x,
*                       fract16 *h, fract16 ny, fract16 nh)
*
* Purpose: convolution of 2 vectors
* In:
*      - x: address of input vector 1 (size ny+nh-1)
*      - h: address of input vector 2 (size nh)
*      - ny: number of element in vector y
*      - nh: number of element in vector h
* Out:
*      - y: address of output vector (size ny)
* In/Out:
*      -
* Read global variables
*      -
* Modified global variables
*      -
* Returns:
*      -
*-------------------------------------------------------------
*/

conv2_q15:

        shr         r0,r0,1
        copyc       lc0,r0                          // ny/2
        setuls0     L_outerLoopSt
        setle0      L_outerLoopEn - 16
        sub         r2,r1,2
        copyc       lc1,r2                          // nh-2
        setuls1     L_innerLoopSt
        setle1      L_innerLoopEn - 16

        sub         r1,r1,1
        copya       p3,r1
        addha       p2,p2,p3                        // h = h + nh - 1
        subba       p12,p3,2

.presliw
        sliwmd
//.align 16
L_outerLoopSt:

// First iteration of inner loop
        ldp         r14,@(p1!+2)                    // load 2 x
        ldh         r15,@(p2!-2)                    // load h
        mpfcll      r12,r14,r15                     // res= (*ph) * (*px)
```

```
        mpfchl      r13,r14,r15                 // res2= (*ph) * (*(px+1))

//.align 16
L_innerLoopSt:
        ldp         r14,@(p1!+2)                // load 2 x
        ldh         r15,@(p2!-2)                // load h
        mafcll      r12,r12,r14,r15             // res+= (*ph) * (*px)
        mafchl      r13,r13,r14,r15             // res2+= (*ph) * (*(px+1))
//.align 8
L_innerLoopEn:

// Last iteration of inner loop
        ldp         r14,@(p1!-p12)              // load 2 x
        ldh         r15,@(p2!+p3)               // load h
        mafcll      r12,r12,r14,r15             // res+= (*ph) * (*px)
        mafchl      r13,r13,r14,r15             // res2+= (*ph) * (*(px+1))

        movehl      r13,r12
        nop
        nop
        sdw         @(p0!+4),r13                // *y++=res, *y++=res2

//.align 8
L_outerLoopEn:
        nop
        nop
        nop
        gp32md

        poprts


        .type       conv2_q15,@function
        .size       conv2_q15,.-conv2_q15
        .globl      __callee.conv2_q15.v.pppii
```

4.11 - cR2FFT_q15_q15

```
/*-------------------------------------------------------------
* FUNCTION : cR2FFT_q15_q15
* Project component : ST100 GENLIB
* File Name : cR2FFT_q15_q15.s
*  Purpose : Radix 2 complex Fast Fourier Transform (decimation in frequency) with
integrated bit-reverse.
*-------------------------------------------------------------
*/

        .text
        .globl          cR2FFT_q15_q15
        .entry32

/*-------------------------------------------------------------
* Function name: void cR2FFT_q15_q15(fract16 *outR, fract16 *outI,
*                        fract16 *inR, fract16 *inI, fract16 *coefOd,
*                        fract16 *coefEv, n)
*
* Purpose :Radix 2 complex Fast Fourier Transform (decimation in frequency) with inte-
grated bit-reverse.
* In :
*      - inR: address of the real input buffer
*      - inI: address of the Imaginary input buffer
*      - coefOd: address of the odds coefficients
*      - coefEv: address of the evens coefficients
*      - n : power of 2 corresponding to the size of the FFT
* Out:
*      - outR: address of real output buffer (align on FFT size boundary)
*      - out: address of imaginary output buffer (align on FFT size boundary)
* In/Out:
*      -
* Read global variables
*      -
* Modified global variables
*      -
* Returns:
*      -
*-------------------------------------------------------------
*/

cR2FFT_q15_q15:
        push    LR2, p4-P8, R4-R11

//*************************************************************
//       FFT: init
//*************************************************************/
        law     p7,@(p15+104)                   // dataI
        law     p8,@(p15+108)                   // Twiddle_even

        law     p5,@(p15+112)                   // Twiddle_odd
        saw     @(p15+104),p0                   // outR

        saw     @(p15+108),p1                   // outI
        setuls0 _fft_stage1_loop_st

        make    R11,1
        shl     R11,R11,R0                      // NINPUTS

        shr     R11,R11,2                       // NINPUTS/4
```

```
        copyc   LC0,R11

        setle0  _fft_stage1_loop_en - 16
        sub     R0,R0,3

        copyc   LC0R,R0                         // LOG2NINPUTS-3
        shl     R11,R11,1

        copya   p4,R11                          // NINPUTS/2
        addba   P1, P5, 0

        addba   P0, p2, 0
        addba   P6, P7, 0

        shl     R0,R11,1
        sub     R0,R0,2

        sub     R0,R0,R11
        copya   P14,R0                          // ((NINPUTS-2)-((NINPUTS)/2))

        ldp     R9, @(P0 + 0)                   // R9H = Real(A1), R9L = Real(B1)

        .presliw
        sliwmd

//*****************************************************************
//       FFT : runtime
//*****************************************************************/

_fft_stage1_loop_st:

        //****************************************
        // this loop is executed
        // NINPUTS/4 times
        //****************************************/

        ldp     R1, @(P0 + p4)      // R1H = Real(B1), R1L = Real(B0)
        ldp     R4, @(P6 + 0)       // R4H = Imag(A1), R4L = Imag(A0)
        addcp   R2, R9, R1          // R2H = Real(A1+B1), R2L = Real(A0+B0)
        subcp   R0, R9, R1          // R0H = Real(A1-B1), R0L = Real(A0-B0)

        ldw     R6, @(P8 !+ 4)      // R6H = Imag(Twiddle1), R6L = Real(Twiddle1)
        mpfchl  R14, R0, R6         // R14 = Real(A1-B1)*Real(Twiddle1)
        mpfhh   R8, R0, R6          // R8 = Real(A1-B1)*Imag(Twiddle1)
        sdp     @(P0 !+ p4), R2     // Real(C0) = Real(A0+B0), Real(C1) = Real(A1+B1)

        ldp     R5, @(P6 + p4)      // R5H = Imag(B1), R4L = Imag(B0)
        ldp     R9, @(P0 - P14)     // R0H = Real(A1), R0L = Real(A0)
        addcp   R3, R4, R5          // R3H = Imag(A1+B1), R3L = Imag(A0+B0)
        subcp   R1, R4, R5          // R1H = Imag(A1-B1), R1L = Imag(A0-B0)

        ldw     R12, @(P1 !+ 4)     // R12H = Imag(Twiddle0), R12L = Real(Twiddle0)
        mpflh   R7, R0, R12         // R7 = Real(A0-B0)*Imag(Twiddle0)
        mpfcll  R13, R0, R12        // R13 = Real(A0-B0)*Real(Twiddle0)
        sdp     @(P6 !+ p4), R3     // Imag(C0) = Imag(A0+B0), Imag(C1)=Imag(A1+B1)

        msfrchh R14, R14, R1, R6    // R14 = R14 - Imag(A1-B1)*Imag(Twiddle1)
        mafrchl R8, R8, R1, R6      // R8 = R8 + Imag(A1-B1)*Real(Twiddle1)
        sdf     @(P0 + 2), R14      // Real(D1) = R14H
        sdf     @(P6 + 2), R8       // Imag(D1) = R8H

        msfrclh R13, R13, R1, R12   // R13 = R13 - Imag(A0-B0)*Imag(Twiddle0)
        mafrcll R7, R7, R1, R12     // R7 = R7 + Imag(A0-B0)*Real(Twiddle0)
```

```
        sdf     @(P0 !- P14), R13         // Real(D0) = R13H
        sdf     @(P6 !- P14), R7          // Imag(D0) = R7H

_fft_stage1_loop_en:

        nop
        shr     R15,R11,3                 // NINPUTS/16
        shra1   p4, p4
        GP32md

        setuls0 _fft_log2_loop_st
        setuls1 _fft_n_1_loop_st

        setle0  _fft_log2_loop_en - 4
        setle1  _fft_n_1_loop_en - 16

        setuls2 _fft_nlog2_loop_st
        shr     R0,R11,2

        copyc   LC1,R0                    // NINPUTS/8
        setle2  _fft_nlog2_loop_en - 16

        shr     R0,R11,3
        copyc   LC1R,R0                   // NINPUTS/16

        shr     R0,R11,1
        copyc   LC0R,R0                   // NINPUTS/4

        makea   P3, 2
        addba   P0, p2, 0

        addba   P6, P7, 0

        .align 16

_fft_log2_loop_st:

        //*****************************************
        // This loop is executed
       // LOG2NINPUTS-3 times
        //*****************************************/

        subba   P3, P3, 1
        shl     R0,R11,1

        sub     R0,R0,2
        copya   P14,R0                    // (NINPUTS-2)

        addba   P1, P5, 0
        makec   LC2, P3

        addba   P3, P3, 1
        subba   P14, P14, p4

        ldp     R9, @(P0 + 0)             // R9H = Real(A1), R9L = Real(B1)

        .presliw
        sliwmd

_fft_n_1_loop_st:

        ldp     R1, @(P0 + p4)            // R1H = Real(B1), R1L = Real(B0)
        ldp     R4, @(P6 + 0)             // R4H = Imag(A1), R4L = Imag(A0)
```

```
        addcp    R2, R9, R1              // R2H = Real(A1+B1), R2L = Real(A0+B0)
        subcp    R0, R9, R1              // R0H = Real(A1-B1), R0L = Real(A0-B0)

        ldp      R6, @(P1 + P3)          // R6H = Imag(Twiddle1), R6L = Real(Twiddle1)
        mpfchl   R14, R0, R6             // R14 = Real(A1-B1)*Real(Twiddle1)
        mpfchh   R8, R0, R6              // R8 = Real(A1-B1)*Imag(Twiddle1)
        sdp      @(P0 !+ p4), R2         // Real(C0)=Real(A0+B0),Real(C1)=Real(A1+B1)

_fft_nlog2_loop_st:

        ldp      R5, @(P6 + p4)          // R5H = Imag(B1), R4L = Imag(B0)
        ldp      R9, @(P0 + p4)          // R0H = Real(A1), R0L = Real(A0)
        addcp    R3, R4, R5              // R3H = Imag(A1+B1), R3L = Imag(A0+B0)
        subcp    R1, R4, R5              // R1H = Imag(A1-B1), R1L = Imag(A0-B0)

        ldp      R12, @(P1 + 0)          // R12H = Imag(Twiddle0), R12L =
Real(Twiddle0)
        mpfclh   R7, R0, R12             // R7 = Real(A0-B0)*Imag(Twiddle0)
        mpfcll   R13, R0, R12            // R13 = Real(A0-B0)*Real(Twiddle0)
        sdp      @(P6 !+ p4), R3         // Imag(C0)=Imag(A0+B0), Imag(C1) =
Imag(A1+B1)

        msfrchh  R14, R14, R1, R6        // R14 = R14 - Imag(A1-B1)*Imag(Twiddle1)
        mafrchl  R8, R8, R1, R6          // R8 = R8 + Imag(A1-B1)*Real(Twiddle1)
        sdf      @(P0 + 2), R14          // Real(D1) = R14H
        sdf      @(P6 + 2), R8           // Imag(D1) = R8H

        msfrclh  R13, R13, R1, R12       // R13=R13 - Imag(A0-B0)*Imag(Twiddle0)
        mafrcll  R7, R7, R1, R12         // R7 = R7 + Imag(A0-B0)*Real(Twiddle0)
        sdf      @(P0 !+ p4), R13        // Real(D0) = R13H
        sdf      @(P6 !+ p4), R7         // Imag(D0) = R7H

        ldp      R1, @(P0 + p4)          // R1H = Real(B1), R1L = Real(B0)
        ldp      R4, @(P6 + 0)           // R4H = Imag(A1), R4L = Imag(A0)
        addcp    R2, R9, R1              // R2H = Real(A1+B1), R2L = Real(A0+B0)
        subcp    R0, R9, R1              // R0H = Real(A1-B1), R0L = Real(A0-B0)

        nop
        mpfchl   R14, R0, R6             // R14 = Real(A1-B1)*Real(Twiddle1)
        mpfchh   R8, R0, R6              // R8 = Real(A1-B1)*Imag(Twiddle1)
        sdp      @(P0 !+ p4), R2         // Real(C0)=Real(A0+B0), Real(C1) =
Real(A1+B1)


_fft_nlog2_loop_en:

        ldp      R5, @(P6 + p4)          // R5H = Imag(B1), R4L = Imag(B0)
        ldp      R9, @(P0 - P14)         // R0H = Real(A1), R0L = Real(A0)
        addcp    R3, R4, R5              // R3H = Imag(A1+B1), R3L = Imag(A0+B0)
        subcp    R1, R4, R5              // R1H = Imag(A1-B1), R1L = Imag(A0-B0)

        mpfclh   R7, R0, R12             // R7 = Real(A0-B0)*Imag(Twiddle0)
        mpfcll   R13, R0, R12            // R13 = Real(A0-B0)*Real(Twiddle0)
        addwa    P1, P1, P3              // P1 updated to point on next twiddle pair
        sdp      @(P6 !+ p4), R3         // Imag(C0)=Imag(A0+B0), Imag(C1) =
Imag(A1+B1)

        msfrchh  R14, R14, R1, R6        // R14=R14 - Imag(A1-B1)*Imag(Twiddle1)
        mafrchl  R8, R8, R1, R6          // R8 = R8 + Imag(A1-B1)*Real(Twiddle1)
        sdf      @(P0 + 2), R14          // Real(D1) = R14H
        sdf      @(P6 + 2), R8           // Imag(D1) = R8H

        msfrclh  R13, R13, R1, R12       // R13=R13 - Imag(A0-B0)*Imag(Twiddle0)
```

```
        mafrcll  R7, R7, R1, R12          // R7 = R7 + Imag(A0-B0)*Real(Twiddle0)
        sdf      @(P0 !- P14), R13        // Real(D0) = R13H
        sdf      @(P6 !- P14), R7         // Imag(D0) = R7H

_fft_n_1_loop_en:

        nop
        shrw     R15, R15, 1
        shra1    p4, p4
        GP32md

        addba    P3, P3, P3
        copyc    LC1R, R15

        addba    P0, p2, 0

        addba    P6, P7, 0

        .align 8
_fft_log2_loop_en:
        shl      r0, r11, 1
        copya    p4, r0                   // p4=bitr(NINPUTS)/2
        bitra    p4, p4
        shra1    p4, p4

        makea    P5, 0
        setuls0  _fft_stagen_1_2_loop_st

        addba    P1, p2, 0
        setle0   _fft_stagen_1_2_loop_en - 4

        law      p0,@(p15+104)            // outR
        law      p6,@(p15+108)            // outI


        ldp      R1, @(P1 + 4)            // R1L = Real(C), R1H = Real(D)
        bitra    P0, P0

        bitra    P6, P6

        .presliw
        sliwmd
_fft_stagen_1_2_loop_st:


        ldp      R12, @(P7 + 4)           // R12L = Imag(C), R12H = Imag(D)
        ldp      R0, @(P1 !+ 8)           // R0L = Real(A), R0H = Real(B)
        addcp    R2, R0, R1               // R2L = Real(A+C), R2H = Real(B+D)
        subcp    R3, R0, R1               // R3L = Real(A-C), R3H = Real(B-D)

        ldp      R0, @(P7 !+ 8)           // R0L = Imag(A), R0H = Imag(B)
        ldp      R1, @(P1 + 4)            // R1L = Real(C), R1H = Real(D)
        subcp    R13, R0, R12             // R13L = Imag(A-C), R13H = Imag(B-D)
        addcp    R12, R0, R12             // R12L = Imag(A+C), R12H = Imag(B+D)

        movell   R3, R13                  // R3L = Imag(A-C), R3H = Real(B-D)
        movell   R13, R3                  // R13L = Real(A-C), R13H = Imag(B-D)
        sdh      %bitr @(P0 !+ P5), R15
        sdf      %bitr @(P6 !+ P5), R14

        nop
        movelh   R2, R12                  // R2L = Real(A+C), R2H = Imag(A+C)
        movehl   R12, R2                  // R12L = Real(B+D), R12H = Imag(B+D)
```

```
        addba   P5, P4, 0

        addcp   R14, R2, R12            // R14L=Real(E)= Real(A+C) + Real(B+D),
                                        // R14H=Imag(E)= Imag(A+C) + Imag(B+D)
        subcp   R15, R2, R12            // R15L=Real(F)= Real(A+C) - Real(B+D),
                                        // R15H=Imag(F)= Imag(A+C) - Imag(B+D)
        sdh     %bitr @(P0 !+ P5), R14  // Real(E) = R14L
        sdf     %bitr @(P6 !+ P5), R14  // Imag(E) = R14H

        movehl  R3, R13                 // R3L = Imag(B-D), R3H = Real(B-D)
        movelh  R13, R3                 // R13L = Real(A-C), R13H = Imag(A-C)
        sdh     %bitr @(P0 !+ P5), R15  // Real(F) = R15L
        sdf     %bitr @(P6 !+ P5), R15  // Real(H) = R15H

        addcp   R14, R13, R3            // R14L=Real(G)= Real(A-C) + Imag(B-D),
                                        // R14H=Imag(H)= Imag(A-C) + Real(B-D)
        subcp   R15, R13, R3            // R15L=Real(H)= Real(A-C) - Imag(B-D),
                                        // R15H-Imag(G)= Imag(A-C) - Real(B-D)
        sdh     %bitr @(P0 !+ P5), R14
        sdf     %bitr @(P6 !+ P5), R15

_fft_stagen_1_2_loop_en:

//*****************************************************************
//       FFT : termination
//*****************************************************************/

        nop
        nop
        sdh     %bitr @(P0 !+ P5), R15
        gp32md

        sdf     %bitr @(P6 !+ P5), R14

        poprts  LR2, p4-P8, R4-R11


        .type   cR2FFT_q15_q15,@function
        .size   cR2FFT_q15_q15,.-cR2FFT_q15_q15
        .globl  __callee.cR2FFT_q15_q15.v.ppppppi
```

4.12 - cR2FFT_q31_q15

```
/*-------------------------------------------------------------
* FUNCTION : cR2FFT_q31_q15
* Project component : ST100 GENLIB
* File Name : cR2FFT_q31_q15.s
* Purpose : In place radix 2 complex Fast Fourier Transform (decimation in frequency).
*-------------------------------------------------------------
*/

        .text
        .globl cR2FFT_q31_q15
        .entry32

/*-------------------------------------------------------------
* Function name: void cR2FFT_q31_q15(fract32 *DataR, fract32 *DataI,
*                        fract16 *coef,int16 n)
*
* Purpose: In place radix 2 complex Fast Fourier Transform (decimation in frequency)
* In:
*      - coef: address of coefficients
*      - n : power of 2 corresponding to the size of the FFT
*
* Out:
*
* In/Out:
*      - dataR : address of the real input and output buffer
*      - dataI: address of the Imaginary input and output buffer
*               (in x space for best performance)
*
* Read global variables
*      -
* Modified global variables
*      -
* Returns:
*      -
*-------------------------------------------------------------
*/
cR2FFT_q31_q15:
        push    R4-R9, P4-P7, LR2

        addwa   p6,p2,0
        sub     r15,r0,3         // Log2(NINPUTS)-3

        make    R9,1
        shl     R9,R9,R0         // NINPUTS

        shr     R0,R9,1          // NINPUTS/2
        copya   P4,R0
        setuls2 __first_stage_ls
        setle2  __first_stage_le-16

        copyc   lc2, R0

        addba   P12, P0, 0       // P12 = dataR
        addba   P14, P1, 0       // P14 = dataI
        subba   P5, P4, 1        // P5 = P4 - 1
        ldw     R3, @(P12+0)     // R3 = dataR[m]
        ldw     R12, @(P14+0)    // R12 = dataI[m]

        .presliw
        sliwmd
```

```
__first_stage_ls:

//      /* butterfly */

        ldw     R1, @(P12+P4)    // R1 = dataR[j]
        ldw     R2, @(P14+P4)    // R2 = dataI[j]
        sub     R13, R3, R1      // R13 = tempr
        sub     R14, R12, R2     // R14 = tempi

        ldw     R0, @(P2 !+ 4)   // R0L = wr, R0H = wi
        NOP
        mpsull  R4, R0, R13      // R4 = temprLL = wr*temprL
        mpsull  R5, R0, R14      // R5 = tempiLL = wr*tempiL

        nop
        nop
        masuhl  R4, R4, R0, R14  // R2 = temprLL = temprLL + wi*tempiL
        mssuhl  R5, R5, R0, R13  // R5 = tempiLL = tempiLL - wi*temprL

        add     R6, R3, R1       // R3 = dataR[m] + dataR[j]
        add     R7, R12, R2      // R12 = dataI[m] + dataI[j]
        sdw     @(P12!+4), R6    // dataR[m] = R6
        sdw     @(P14!+4), R7    // dataR[m] = R7

        ldw     R3, @(P12+0)     // R3 = dataR[m+1]
        ldw     R12, @(P14+0)    // R12 = dataI[m+1]
        shr     R4, R4, 15       // R2 = temprLL >>= 15
        shr     R5, R5, 15       // R5 = tempiLL >>= 15

        nop
        nop
        maflh   R4, R4, R0, R13  // R1 = temprLH = (temprLL >> 15) + wr*temprH
        maflh   R5, R5, R0, R14  // R4 = tempiLH = (tempiLL >> 15) +wr*tempiH

        mafhh   R4, R4, R0, R14  // R1 = temprLH = temprLH + wi*tempiH
        msfhh   R5, R5, R0, R13  // R3 = tempiLH = tempiLH - wi*temprH

        sdw     @(P12 + P5), R4  // dataR[j] = temprLH
        sdw     @(P14 + P5), R5  // dataI[j] = tempiLH

__first_stage_le:

        nop
        nop
        makea   P3, 2            // P3 = wDist2
        GP32md

        setuls0 __log_n_loop_st
        setle0  __log_n_loop_en - 4
        copyc   lc0, R15         // log2(N)-3
        setuls1 __n_loop_st
        setle1  __n_loop_en - 16
        setuls2 __n_log_n_loop_st
        setle2  __n_log_n_loop_en - 16

        shr     r0,r9,2          // NINPUTS/4
        copya   p4,r0

        .align 16
__log_n_loop_st:

  /* total number of passes NP: log2(N)-3 */
```

```
        addba   P12, P3, 0       // P12 = P3
        addba   P3, P4, 0        // P3 = mmax/2 = P4

        makec   lc1, P3
        subba   P3, P12, 1

        makec   lc2, P3
        addba   P3, P3, 1        // P3 = wDist2

        addba   P12, P0, 0       // P12 = dataR

        sub     R0,R9,1
        copya   p5,r0            // NINPUTS-1


        addba   P14, P1, 0       // P14 = dataI

        addwa   p2,p6,0
        subba   P5, P4, P5       // P5 = -(NINPUTS-1) + P4

        .presliw

        ldw     R3, @(P12+0)     // R3 = ptrDataR[0]
        ldw     R12, @(P14+0)    // R12 = ptrDataI[0]
        nop
        nop

        sliwmd
__n_loop_st:

        /* Butterfly loop prolog */

        ldw     R1, @(P12+P4)    // R1 = ptrDataR[mmax_div2]
        ldw     R2, @(P14+P4)    // R2 = ptrDataI[mmax_div2]
        sub     R13, R3, R1      // R13 = tempr
        sub     R14, R12, R2     // R14 = tempi

        ldw     R0, @(P2!+P3)    // R0 = *wp, wp+=wDist2
        nop
        mpsull  R4, R0, R13      // R2 = temprLL = wr*temprL
        mpsull  R5, R0, R14      // R12 = tempiLL = wr*tempiL

__n_log_n_loop_st:


        nop
        nop
        masuhl  R4, R4, R0, R14  // R2 = temprLL = temprLL + wi*tempiL
        mssuhl  R5, R5, R0, R13  // R5 = tempiLL = tempiLL - wi*temprL

        add     R7, R12, R2      // R5 = ptrDataI[0] + ptrDataI[mmax_div2]
        add     R6, R3, R1       // R4 = ptrDataR[0] + ptrDataR[mmax_div2]
        sdw     @(P12!+P4), R6   // ptrDataR[0] = R6, ptrDataR+=mmax_div2
        sdw     @(P14!+P4), R7   // ptrDataI[0] = R7, ptrDataI+=mmax_div2

        nop
        nop
        shr     R4, R4, 15       // R2 = temprLL >>= 15
        shr     R5, R5, 15       // R12 = tempiLL >>= 15

        ldw     R3, @(P12+P4)    // R3 = ptrDataR[mmax_div2]
        ldw     R12, @(P14+P4)   // R12 = ptrDataI[mmax_div2]
        maflh   R4, R4, R0, R13  // R1 = temprLH = wr*temprH
```

```
        maflh   R5, R5, R0, R14  // R3 = tempiLH = wr*tempiH


        mafhh   R4, R4, R0, R14  // R1 = temprLH = temprLH + wi*tempiH
        msfhh   R5, R5, R0, R13  // R3 = tempiLH = tempiLH - wi*temprH
        sdw     @(P12 !+ P4), R4 // ptrDataR[0] = R2, ptrDataR+=mmax_div2
        sdw     @(P14 !+ P4), R5 // ptrDataI[0] = R12, ptrDataI+=mmax_div2

        /* butterfly */

        ldw     R1, @(P12+P4)    // R1 = ptrDataR[mmax_div2]
        ldw     R2, @(P14+P4)    // R2 = ptrDataI[mmax_div2]
        sub     R13, R3, R1      // R13 = tempr
        sub     R14, R12, R2     // R14 = tempi

        nop
        nop
        mpsull  R4, R0, R13      // R2 = temprLL = wr*temprL
        mpsull  R5, R0, R14      // R12 = tempiLL = wr*tempiL

__n_log_n_loop_en:

        nop
        nop
        masuhl  R4, R4, R0, R14  // R2 = temprLL = temprLL + wi*tempiL
        mssuhl  R5, R5, R0, R13  // R5 = tempiLL = tempiLL - wi*temprL

        add     R7, R12, R2      // R5 = ptrDataI[0] + ptrDataI[mmax_div2]
        add     R6, R3, R1       // R4 = ptrDataR[0] + ptrDataR[mmax_div2]
        sdw     @(P12!+P4), R6   // ptrDataR[0] = R6, ptrDataR+=mmax_div2
        sdw     @(P14!+P4), R7   // ptrDataI[0] = R7, ptrDataI+=mmax_div2

        nop
        nop
        shr     R4, R4, 15       // R2 = temprLL >>= 15
        shr     R5, R5, 15       // R12 = tempiLL >>= 15

        ldw     R3, @(P12+P5)    // R3 = ptrDataR[-NINPUTS+mmax_div2+1]
        ldw     R12, @(P14+P5)   // R12 = ptrDataI[-NINPUTS+mmax_div2+1]
        maflh   R4, R4, R0, R13  // R1 = temprLH = wr*temprH
        maflh   R5, R5, R0, R14  // R3 = tempiLH = wr*tempiH


        mafhh   R4, R4, R0, R14  // R1 = temprLH = temprLH + wi*tempiH
        msfhh   R5, R5, R0, R13  // R3 = tempiLH = tempiLH - wi*temprH
        sdw     @(P12 !+ P5), R4 // ptrDataR[0] = R4, P12 = &dataR[i]
        sdw     @(P14 !+ P5), R5 // ptrDataI[0] = R5, P14 = &dataI[i]


__n_loop_en:

        nop
        nop
        shra1   P4, P4           // P4 = mmax = mmax/2
        GP32md

        addba   P3, P3, P3       // P3 = wDist2 * 2
        .align 8
__log_n_loop_en:

     /* last two stages */
        setuls0  __last_2_stages_st
        setle0   __last_2_stages_en - 4
```

```
        shr     R0,R9,2
        copyc   lc0,R0          // NINPUTS/4
        addba   P12, P0, 0      // P12 = dataR
        addba   P14, P1, 0      // P14 = dataI
        ldw     R8, @(P12 + 0)  // R8 = dataR[i]
        .presliw
        sliwmd

__last_2_stages_st:

        ldw     R1, @(P12 + 8)  // R1 = dataR[i+2]
        ldw     R2, @(P14 + 0)  // R2 = dataI[i]
        sub     R12, R8, R1     // R12 = temprA = dataR[i] - dataR[i+2]
        add     R13, R8, R1     // R13 = temprB = dataR[i] + dataR[i+2]

        ldw     R3, @(P14 + 8)  // R3 = dataI[i+2]
        ldw     R0, @(P12 + 4)  // R0 = dataR[i+1]
        sub     R14, R2, R3     // R14 = tempiA = dataI[i] - dataI[i+2]
        add     R15, R2, R3     // R15 = tempiB = dataI[i] + dataI[i+2]

        /* butterfly */

        ldw     R1, @(P12 + 12) // R1 = dataR[i+3]
        ldw     R2, @(P14 + 4)  // R2 = dataI[i+1]
        sub     R4, R0, R1      // R4 = temprD = dataR[i+1] - dataR[i+3]
        add     R5, R0, R1      // R5 = temprC = dataR[i+1] + dataR[i+3]

        ldw     R8, @(P12 + 16) // R0 = dataR[i+4]
        ldw     R3, @(P14 + 12) // R3 = dataI[i+3]
        sub     R0, R2, R3      // R0 = tempiD = dataI[i+1] - dataI[i+3]
        add     R1, R2, R3      // R1 = tempiC = dataI[i+1] + dataI[i+3]

        /* butterfly */

        add     R6, R13, R5     // R6 = temprB + temprC
        add     R7, R15, R1     // R7 = tempiB + tempiC
        sdw     @(P12 !+ 16), R6 // dataR[i] = R6, dataR+=4
        sdw     @(P14 !+ 16), R7 // dataI[i] = R7, dataI+=4

        sub     R6, R13, R5     // R6 = temprB - temprC
        sub     R7, R15, R1     // R7 = tempiB - tempiC
        sdw     @(P12 - 12), R6 // dataR[i+1] = R6
        sdw     @(P14 - 12), R7 // dataI[i+1] = R7

        /* butterfly */

        add     R6, R12, R0     // R6 = temprA + tempiD
        sub     R7, R14, R4     // R7 = tempiA - temprD
        sdw     @(P12 - 8), R6  // dataR[i+2] = R6
        sdw     @(P14 - 8), R7  // dataI[i+2] = R7

        sub     R6, R12, R0     // R6 = temprA - tempiD
        add     R7, R14, R4     // R7 = tempiA + temprD
        sdw     @(P12 - 4), R6  // dataR[i+3] = R6
        sdw     @(P14 - 4), R7  // dataI[i+3] = R7

__last_2_stages_en:

        nop
        nop
        nop
        GP32md
```

```
poprts   R4-R9, P4-P7, LR2

.type    cR2FFT_q31_q15,@function
.size    cR2FFT_q31_q15,.-cR2FFT_q31_q15
.globl   __callee.cR2FFT_q31_q15.v.pppi
```

4.13 - FIR_q15

```
/*-------------------------------------------------------------
* FUNCTION : FIR_q15
* Project component : ST100 GENLIB
* File Name : FIR_q15.s
* Purpose : Real FIR filter (direct form) of a 16-bit fractional vector.
*-------------------------------------------------------------
*/

         .define DELAYMOD 128

        .text
         .entry32
         .globl FIR_q15
         .align 16

/*-------------------------------------------------------------
* Function name : void FIR_q15(fract16 *out, fract16 *in, fract16 *coefs,
*                     fract16 **delay, int16 nIn, int16 nCoefs, int16 scale);
* Author :
* Purpose :Real FIR filter (direct form) of a 16-bit fractional vector.
* In :
*      - in address of the input buffer
*      - coefs address of the coefficients
*      - delay pointer on the address of a delay line (size=64 bytes)
*      - nIn number of sample in input buffer
*      - nCoefs number of coefficients (maxi=16)
*      - scale scale bit for each output
* Out :
*      - out address of output buffer
* In/Out:
*      -
* Read global variables
*      -
* Modified global variables
*      -
* Returns:
*      -
*-------------------------------------------------------------
*/
FIR_q15:
         push     LR1,p4-p5
         shr      R0,R0,1

         copyc    LC0, R0
         setuls0 _block_loop_start

         law      P3, @(P15+24)
         law      P14,@(P3+0)

         setle0   _block_loop_end - 16
         sub      R1,R1,1

         copya    P4, R1
         sub      R1,R1,2

         copyc    LC1, R1
         addba    P12, P4, 2

         setuls1  _filter_loop_start
         setle1   _filter_loop_end - 16
```

```
        .presliw
        sliwmd


_block_loop_start:

        // This loop is executed nIn/2 times

        // first iteration of computation is done here
        // therefore it prevents from initialization of sum1 and sum2 at 0
        ldw     R3, @(P2 !+ 4)                // load two coefs
       ldhsw    R12, @(P1 !+ 4)               // load two samples, and swapped it
        mpfcll  R0, R12, R3                   // coefs[0] * sample[n+1] -> R0
        mpfchl  R1, R12, R3                   // coefs[0] * sample[n] -> R1

        // second iteration of computation is done here
        // therefore it avoids a memory hit
        ldlh    %DELAYMOD R13, @(P14 + 4)     // delay_line: sample[n-1] -> R13
        mafchh  R0, R0, R12, R3               // sample[n] * coefs[1] + R0 -> R0
        mafclh  R1, R1, R13, R3               // coefs[1] * sample[n-1] + R1 -> R1
        sdw     %DELAYMOD @(P14 !+ 4), R12    //sample[n],sample[n-1] -> delayline

_filter_loop_start:

        // This loop is executed NCOEFS-3 times
        ldlh    R3, @(P2 !+ 2)                // load coefs[i]
        ldp     %DELAYMOD R13, @(P14 !+ 2)    // load sample[p], sample[p+1]
        mafcll  R0, R0, R13, R3               // coefs[i] * sample[p+1] + R0 -> RO
        mafchl  R1, R1, R13, R3               // coefs[i] * sample[p] + R1 -> R1

_filter_loop_end:

        // last iteration of computation is done here
        // therefore using pointer post decrementation facilities PTRcoefs
        // will be updated for next _block_loop iteration
        ldlh    R3, @(P2 !- P4)               // PTRcoefs update
        ldp     %DELAYMOD R13, @(P14 !- P12)  // Delay line update
        mafcll  R0, R0, R13, R3
        mafchl  R1, R1, R13, R3

        // It will take 2 cycles to execute the above instructions
        // due to MAC operator latency

        shr     R0, R0, R2
        shr     R1, R1, R2
        sdf     @(P0 !+ 4), R1                // output[n] at @P0
        sdf     @(P0 + 2), R0                 // output[n+1] at @(P0 + 2)

        // PTRdelay equals PTRdelay at the beginning of the _block_loop
        // minus 2 modulo NDELAY2

_block_loop_end:

        nop
        nop
        saw     @(P3+0), P14                  // return next delay line address
        gp32md

        poprts  LR1,p4-p5

        .type   FIR_q15,@function
        .size   FIR_q15,.-FIR_q15

        .globl  __callee.FIR_q15.v.ppppiii
```

4.14 - i16_maxValGP16_16

```
/*-------------------------------------------------------------
* FUNCTION : i16_maxValGP16_16
* Project component : ST100 GENLIB
* File Name : i16_maxValGP16_16.s
* Purpose : Return the maximum element of an integer or fractional 16-bit vector.
* GP16 instruction set
*-------------------------------------------------------------
*/

        .text
        .entry16
        .globl          i16_maxValGP16_16

/*-------------------------------------------------------------
* Function name : int16 r i16_maxValGP16_16 (int16 *i, int16 ni)
*
* Purpose: Return the maximum element of an integer or fractional 16-bit vector.
* In:
*      - i Address of the vector
*      - ni Size of the vector
* Out:
*      -
* In/Out:
*      -
* Read global variables
*      -
* Modified global variables
*      -
* Returns:
*      - r Maximum element
*
* In order to decrease code size, we use GP16 code.
* function uses only scratch registers. No need push and pop.
* Search of maximum is computed inside a software loop (labels .l2 and .l1).
* The loop is executed N times and r0 is used as the loop counter.
* At each step of the loop, loaded value is compared with the present
* maximum value placed in r2. If it is greater, it is moved into r2.
*-------------------------------------------------------------
*/

i16_maxValGP16_16:

g0?     makesr  0b00000100000

g0?     moresr  0b0000000000
g0?     exth    r2,r3

        goto .l1

 .l2:
 g0?    ldh     r1,@(p0)        // loads the next value in the array
 g0?    decu    r0,1            // decrements the loop counter

 g0?    exth    r1,r1
 g0?    addba   p0,p0,2         // increments the address pointer on the array

        gtw     g0,r1,r2        // loaded value compared with the present maximum
                                // value.
 g0?    move    r2,r1           // loaded value moved in r2 if it is greater than the
                                // present maximum value.
```

```
.l1:
        eqw     g0,r0,0          // Tests if the loop has been executed N times
g0!     goto    .l2
g0?     move    r0,r2

        rts

        .type   i16_maxValGP16_16,@function
        .size   i16_maxValGP16_16,.-i16_maxValGP16_16
        .globl  __callee.i16_maxValGP16_16.i.pi
```

4.15 - i16_maxValSLIW_16

```
/*-------------------------------------------------------------
* FUNCTION : i16_maxValSLIW_16
* Project component : ST100 GENLIB
* File Name : i16_maxValSLIW_16.s
* Purpose : Return the maximum element of an integer or fractional 16-bit vector.
* SLIW mode
*-------------------------------------------------------------
*/

        .text
        .entry32
        .globl          i16_maxValSLIW_16

/*-------------------------------------------------------------
* Function name: int16 r i16_maxValSLIW_16 (int16 *i, int16 ni)
*
* Purpose: Return the maximum element of an integer or fractional 16-bit vector.
* In:
*      - i Address of the vector
*      - ni Size of the vector (must be a multiple of 4)
* Out:
*      -
* In/Out:
*      -
* Read global variables
*      -
* Modified global variables
*      -
* Returns:
*      - r Minimum element
*
* function uses only scratch registers. No need push and pop.
* To search the maximum, the array is divided into 2.
* At the beginning, one pointer (p0) addresses tab[0], the other (p1)addresses tab[N/2]
* Then, in the loop, the minimum value is searched in each part of the array. This is
* for using both ST100 unit.
* In each part, maximum value is searched using packed instructions (array coefficients
* are short) in order to save cycles. Temporary maxima values are packed in r2 for the
* lower addressees of the array, in r3 for the upper addresses of the array.
* Loop iteration is equal to N/4
* At the end of the loop, the code compares the 4 maxima value got in r2 and r3
* and moves the maximum value in r0,the return parameter.
*-------------------------------------------------------------
*/

i16_maxValSLIW_16:

        shr     r1,r0,1      /* computes address for the second pointer */
        SETULS0 debutL0

        copya   p2,r1
        SETLE0  (finL0-4)

        shr     r4,r0,2
        makep   r2,0x8000    /* initialization of the reference registers */
                             /* with the smallest short value */
        makep   r3,0x8000
        copyc   LC0,r4

        addha   p1,p0,p2     /* initializes the second pointer at*/
                             /* the middle address of the array */
```

```
        .presliw
        sliwmd


debutL0:        ldw     r12,@(p0!+4)  /* loads 2 short data in the first part
                                         of the array */
                ldw     r13,@(p1!+4)  /* loads 2 short data in the second part
                                         of the array */
                gtp     g0,r12,r2     /* packed comparison between the present
                                         values and the greater found up to now
                                         in the first part of the array */
                gtp     g1,r13,r3     /* idem with the second part */

        g0?     addp    r2,r12,0      /* if one or both short data in r12
                                        greater than one or both in r2, it
                                         or they are saved in r2.*/
        g1?     addp    r3,r13,0      /* idem with the second part of the array */
                nop
                nop


finL0:


                gtp     g0,r3,r2    /* comparison between the 4 greatest values
                                       found in the array */
                nop
                nop
                gp32md

        g0?     addp    r2,r3,0     /* saves the 2 greatest values found in the array*/
                movehl  r12,r2      /* Then these 2 greatest values are compared */

                gtp     g1,r12,r2
        g1?     addp    r2,r12,0    /* r2 saves the maximum */

                exth    r0,r2       /* maximum returned in r0 */
                rts

        .type   i16_maxValSLIW_16,@function
        .size   i16_maxValSLIW_16,.-i16_maxValSLIW_16

        .globl  __callee.i16_maxValSLIW_16.i.pi


/*---------------------------------------------------------------
```

4.16 - i32_maxValGP16_32

```
/*-------------------------------------------------------------
* FUNCTION : i32_maxValGP16_32
* Project component : ST100 GENLIB
* File Name : i32_maxValGP16_32.s
* Purpose : Return the maximum element of an integer or fractional 32-bit vector.
* GP16 instruction set
*-------------------------------------------------------------
*/

         .text
         .entry16
         .globl           i32_maxValGP16_32

/*-------------------------------------------------------------
* Function name: int32 r i32_maxValGP16_32 (int32 *i, int16 ni)
*
* Purpose: Return the maximum element of an integer or fractional 32-bit vector.
* In:
*      - i Address of the vector
*      - ni Size of the vector
* Out:
*      -
* In/Out:
*      -
* Read global variables
*      -
* Modified global variables
*      -
* Returns:
*      - r Maximum element
*
* To increase code density, GP16 code is used.
* function uses only scratch registers. No need of push and pop.
*-------------------------------------------------------------
*/
i32_maxValGP16_32:

g0?      makesr           0b00000000001

g0?      shlu     r3,0b11111
g0?      extw     r2,r3

         goto .l1

 .l2:
 g0?     lduw     r1,@(p0)
 g0?     decu     r0,1

 g0?     extw     r1,r1
 g0?     addba    p0,p0,4

         gtw      g0,r1,r2
 g0?     move     r2,r1

 .l1:
         eqw      g0,r0,0
 g0!     goto     .l2
 g0?     move     r0,r2

         rts
```

```
.type    i32_maxValGP16_32,@function
.size    i32_maxValGP16_32,.-i32_maxValGP16_32
.globl   __callee.i32_maxValGP16_32.i.pi
```

4.17 - i32_maxValSLIW_32

```
/*-------------------------------------------------------------
* FUNCTION : i32_maxValSLIW_32
* Project component : ST100 GENLIB
* File Name : i32_maxValSLIW_32.s
* Purpose : Return the maximum element of an integer or fractional 32-bit vector.
* SLIW mode
*-------------------------------------------------------------
*/


         .text
         .entry32
         .globl           i32_maxValSLIW_32

/*-------------------------------------------------------------
* Function name: int32 r i32_maxValSLIW_32 (int32 *i, int16 ni)
* Purpose:  Return the maximum element of an integer or fractional 32-bit vector.
* In:
*      - i Address of the vector
*      - ni Size of the vector (must be a multiple of 2)
* Out:
*      -
* In/Out:
*      -
* Read global variables
*      -
* Modified global variables
*      -
* Returns:
*      - r Maximum element
*
* function uses only scratch registers. No need for push and pop.
* *-------------------------------------------------------------
*/

i32_maxValSLIW_32:

          shr       r1,r0,1
          SETULS0   debutL0

          copya     p2,r1
          SETLE0    (finL0-4)


          make      r2,0x8000

          copyc     LC0,r1

          make      r3,0x8000
          addwa     p1,p0,p2

          more      r2,0x0000
          more      r3,0x0000

          .presliw
          sliwmd
```

```
debutL0:        ldw     r12,@(p0!+4)

                ldw     r13,@(p1!+4)

                gtw     g0,r12,r2


                gtw     g1,r13,r3

        g0?     add     r2,r12,0


        g1?     add     r3,r13,0
                nop
                nop



finL0:


                gtw     g0,r3,r2

                nop
                nop
                gp32md

        g0?     add     r2,r3,0
                add     r0,r2,0

                rts

        .type   i32_maxValSLIW_32,@function
        .size   i32_maxValSLIW_32,.-i32_maxValSLIW_32

        .globl  __callee.i32_maxValSLIW_32.i.pi
```

4.18 - i16_minValGP16_16

```
/*-------------------------------------------------------------
* FUNCTION : i16_minValGP16_16
* Project component : ST100 GENLIB
* File Name : i16_minValGP16_16.s
* Purpose : Return the minimum element of an integer or fractional 16-bit vector.
* GP16 instruction set
*-------------------------------------------------------------
*/


         .text
         .entry16
         .globl           i16_minValGP16_16


/*-------------------------------------------------------------
* Function name: int16 r i16_minValGP16_16 (int16 *i, int16 ni)
*
* Purpose: Return the minimum element of an integer or fractional 16-bit vector.
* In:
*      - i Address of the vector
*      - ni Size of the vector
* Out:
*      -
* In/Out:
*      -
* Read global variables
*      -
* Modified global variables
*      -
* Returns:
*      - r Minimum element
*
* In order to decrease code size, we use GP16 code.
* function uses only scratch registers. No need push and pop.
*-------------------------------------------------------------
*/

i16_minValGP16_16:

g0?      makesr   0b00000011111
g0?      moresr   0b1111111111
g0?      exth     r2,r3


         goto .l1

 .l2:
 g0?     ldh      r1,@(p0)
 g0?     decu     r0,1

 g0?     exth     r1,r1
 g0?     addba    p0,p0,2

         gew      g0,r1,r2
         notg     g0

 g0?     move     r2,r1

 .l1:
```

```
        eqw     g0,r0,0
g0!     goto    .l2
g0?     move    r0,r2

        rts

        .type   i16_minValGP16_16,@function
        .size   i16_minValGP16_16,.-i16_minValGP16_16
        .globl  __callee.i16_minValGP16_16.i.pi
```

4.19 - i16_minValSLIW_16

```
/*-------------------------------------------------------------
* FUNCTION : i16_minValSLIW_16
* Project component : ST100 GENLIB
* File Name : i16_minValSLIW_16.s
* Purpose : Return the minimum element of an integer or fractional 16-bit vector.
* SLIW mode
*-------------------------------------------------------------
*/

        .text
        .entry32
        .globl          i16_minValSLIW_16
/*-------------------------------------------------------------
* Function name: int16 r i16_minValSLIW_16 (int16 *i, int16 ni)
*
* Purpose: Return the minimum element of an integer or fractional 16-bit vector.
* In:
*      - i Address of the vector
*      - ni Size of the vector (must be a multiple of 4)
* Out:
*      -
* In/Out:
*      -
* Read global variables
*      -
* Modified global variables
*      -
* Returns:
*      - r Minimum element
*
* Function only uses scratch registers. No need for push and pop.
* To search the minimum, the array is divided into 2.
* At the beginning, one pointer (p0) addressees tab[0], the other (p1)adresses tab[N/2]
* Then, in the loop, the minimum value is searched in each part of the array. This allows
* to use the two ST100 units.
* In each part, minimum value is searched using packed instructions (array coefficients
* are short) in order to save cycles. Temporary minima values are packed in r2 for the
* lower addressees of the array, in r3 for the upper addresses of the array.
* Loop iteration is equal to N/4
* At the end of the loop, the code compares the 4 minima values placed in r2 and r3
* and moves the minimum value in r0,the return parameter.
*-------------------------------------------------------------
*/

i16_minValSLIW_16:
          shr       r3,r0,2
          copyc     LC0,r3
          SETULS0   debutL0
          SETLE0    (finL0-4)
          makep     r2,0x7fff       // initialization of the reference registers
                                    // with the smallest short value shr r1,r0,1
                                    // computes address for the second pointer
          makep     r3,0x7fff
          copya     p2,r1
          addha     p1,p0,p2        // initializes the second pointer at
                                    // the middle address of the array
          .presliw
          sliwmd
 debutL0: ldw       r12,@(p0!+4)    // loads 2 short data in the first part
                                    // of the array
          ldw       r13,@(p1!+4)    // loads 2 short data in the second part
                                    // of the array
```

```
        ltp         g0,r12,r2     // packed comparison between the present
                                  // values and the greater found up to now
                                  // in the first part of the array
        ltp         g1,r13,r3     // idem with the second part
  g0?   addp        r2,r12,0      // if one or both short data in r12
                                  // greater than one or both in r2, it
                                  // or they are saved in r2.
  g1?   addp        r3,r13,0      // idem with the second part of the array
        nop
        nop
finL0:  ltp         g0,r3,r2      // comparison between the 4 greatest values
                                  // found in the array
        nop
        nop
        gp32md

  g0?   addp        r2,r3,0       // saves the 2 greatest values found in the array
  movehl r12,r2                   // Then these 2 greatest values are compared
        ltp         g1,r12,r2
  g1?   addp        r2,r12,0      // r2 saves the maximum
        exth        r0,r2         // maximum returned in r0
         rts


.type   16_minValSLIW_16,@function
.size   i16_minValSLIW_16,.-i16_minValSLIW_16
.globl  __callee.i16_minValSLIW_16.i.pi
```

4.20 - i32_minValGP16_32

```
/*-------------------------------------------------------------
* FUNCTION : i32_minValGP16_32
* Project component : ST100 GENLIB
* File Name : i32_minValGP16_32.s
* Purpose :  Return the minimum element of an integer or fractional 32-bit vector.
* GP16 instruction set
*-------------------------------------------------------------
*/
        .text
        .entry16
        .globl   i32_minValGP16_32
/*-------------------------------------------------------------
* Function name : int32 r i32_minValGP16_32 (int32 *i, int16 ni)
* Purpose :  Return the minimum element of an integer or fractional 32-bit vector
* In :
*      - i Adress of the vector
*      - ni Size of the vector
* Out :
*      -
* In/Out :
*      -
* Read global variables
*      -
* Modified global variables
*      -
* Returns :
*      - r Minimum element
*
* To increase code density , GP16 code is used.
* function uses  scratch registers only. No need push and pop.
*-------------------------------------------------------------
*/


i32_minValGP16_32:



//function uses  scratch registers only. No need push and pop


g0?     makesr   0b11111111110
g0?     extw     r2,r3
g0?     makesr   0x1
g0?     rotlw    r2,r3

        goto .l1

 .l2:

 g0?    lduw     r1,@(p0)
 g0?    decu     r0,1
 g0?    extw     r1,r1
 g0?    addba    p0,p0,4

        gew      g0,r1,r2
        notg     g0

 g0?    move     r2,r1
```

```
.l1:

        eqw     g0,r0,0
g0!     goto    .l2
g0?     move    r0,r2
        rts
        .type   i32_minValGP16_32,@function
        .size   i32_minValGP16_32,.-i32_minValGP16_32
        .globl  __callee.i32_minValGP16_32.i.pi
```

4.21 - i32_minValSLIW_32

```
/*-------------------------------------------------------------
* FUNCTION : i32_minValSLIW_32
* Project component : ST100 GENLIB
* File Name : i32_minValSLIW_32.s
* Purpose :  Return the minimum element of an integer or fractional 32-bit vector.
* SLIW mode
*-------------------------------------------------------------
*/
         .text
         .entry32
         .globl   i32_minValSLIW_32
/*-------------------------------------------------------------
* Function name : int32 r i32_minValSLIW_32 (int32 *i, int16 ni)
* Purpose :  Return the minimum element of an integer or fractional 32-bit vector.
* In :
*      - i Adress of the vector
*      - ni Size of the vector (must be a multiple of 2)
* Out :
*      -


* In/Out :
*      -
* Read global variables
*      -

* Modified global variables
*      -
* Returns :
*      - r Minimum element
*
* function uses  scratch registers only. No need push and pop.
* *-------------------------------------------------------------
*/

i32_minValSLIW_32:

         shr      r1,r0,1
         SETULS0  debutL0

         copya    p2,r1
         SETLE0   (finL0-4)

         make     r2,0x7fff
         copyc    LC0,r1

         make     r3,0x7fff
         addwa   p1,p0,p2

         more     r2,0xffff
         more     r3,0xffff

         .presliw
         sliwmd

 debutL0:        ldw      r12,@(p0!+4)
                 ldw      r13,@(p1!+4)
                 ltw      g0,r12,r2
                 ltw      g1,r13,r3
```

```
        g0?     add     r2,r12,0

        g1?     add     r3,r13,0
                nop
                nop

finL0:
                ltw     g0,r3,r2
                nop
                nop
                gp32md

        g0?     add     r2,r3,0
                add     r0,r2,0
                rts


        .type   i32_minValSLIW_32,@function
        .size   i32_minValSLIW_32,.-i32_minValSLIW_32
        .globl  __callee.i32_minValSLIW_32.i.pi
```

4.22 - IIRBiquad4_q15

```
/*-------------------------------------------------------------
* FUNCTION : IIRBiquad4_q15
* Project component : ST100 GENLIB
* File Name : IIRBiquad4_q15.s
* Purpose : IIR real filter with n cascaded biquads of 4 coefficients.
*-------------------------------------------------------------
*/


        .text
        .globl IIRBiquad4_q15
        .entry32
        .align 16

/*-------------------------------------------------------------
* Function name : void IIRBiquad4_q15(fract16 *out, fract16 *in, fract16 *coefs,
*                    fract16 *delay, int16 n, int16 nb);
*
* Purpose:IIR real filter with n cascaded biquads of 4 coefficients.
* In:
*      - in address of the input buffer
*      - coefs address of the coefficients
*      - delay address of a delay line
*      - n number of sample in input and output buffer
*      - nb number of biquad
*
* Out:
*      - out address of output buffer
* In/Out:
*      -
* Read global variables
*      -
* Modified global variables
*      -
* Returns:
*      -
*-------------------------------------------------------------
*/

IIRBiquad4_q15:

        copyc   LC0,R0                          // Number of samples
        setuls0 iirOuterStart

        setle0  iirOuterEnd - 16
        copyc   LC1,R1                          // Number of biquads

        setuls1 iirInnerStart
        setle1  iirInnerEnd - 16

        law     P3,@(P15+0)                     // pointer on delayline
        addba   p12,p2,0

        addba   p14,p3,0
        ldh     r3,@(p1!+2)


        .presliw
        SLIWmd
//.align 16
```

```
iirOuterStart:

        shl     r0,r3,16
        nop
        addba   p3,p14,0                // save pointer on Delay line
        addba   p2,p12,0                // save pointer on coefficients

iirInnerStart:

        ldw     r2, @(p3 + 0)           // r2L = Delay[0], r2H = Delay[1]
        ldw     r12, @(p2 !+ 4)         // r12L = Coef[2], r12H = Coef[0]
        mafcll  r1, r0, r2, r12         // r1 = t0 = x+Delay[0]*Coef[2]
        mpfclh  r13, r2, r12            // r13 = x0 = Delay[0]*Coef[0]

        ldw     r12, @(p2 !+ 4)         // r12L = Coef[3], r12H = Coef[1]
        mafchl  r1, r1, r2, r12         // r1=t0=t0+Delay[1]*Coef[3]
        mafchh  r13, r13, r2, r12       // r13 = x0 = x0+Delay[1]*Coef[1]
        sdh     @(p3 + 2), r2           // Delay[1]= Delay[0]

        addcw   r0,r1,r13               // y ready
        nop
        nop
        sdf     @(p3!+4),r1             // Delay[0] = t0


iirInnerEnd:

        ldh     r3,@(p1!+2)
        nop
        nop
        sdf     @(p0!+2),r0             // store output

iirOuterEnd:

        nop
        nop
        nop
        gp32md


        rts

.type   IIRBiquad4_q15,@function
.size   IIRBiquad4_q15,.-IIRBiquad4_q15

.globl  __callee.IIRBiquad4_q15.v.ppppii
```

4.23 - mMul_16_16_16

```
/*-------------------------------------------------------------
* FUNCTION : mMul_16_16_16
* Project component : ST100 GENLIB
* File Name : mMul_16_16_16.s
* Purpose : Matrix multiply 16-bit integer, result on 16-bit integer.
* GP32 instruction set
*-------------------------------------------------------------
*/

        .text
        .entry32
        .globl          mMul_16_16_16

.align 8
/*-------------------------------------------------------------
* Function name: void mMul_16_16_16 (int16 *matres, int16 *mat1,int16 *mat2, int16
row1,int16 col2,int16 col1)
*
* Purpose:  Matrix multiply 16-bit integer, result on 16-bit integer(16b x 16b -> 16b).
* In:
*      - mat1 address of the matrix1
*      - mat2 address of the matrix2
*      - row1 number of rows of matrix1
*      - col2 number of columns of matrix2
*      - col1 number of columns of matrix1
* Out:
*      - matres address of result matrix
* In/Out:
*      -
* Read global variables
*      -
* Modified global variables
*      -
* Returns:
*      -
*-------------------------------------------------------------
*/
mMul_16_16_16:

        push    r4-r5,p4-p5,lr1,lr2  // saves used registers
        mpuull  r4,r1,r2             // prepares N=(pm-1) used for pointer manipula-
tions
                                     // on the second matrix (column loop)

        tbpos   g1,r2,0              // test of the parity of p to prepare word loading
                                     // g1 =1 if p is odd
        SETULS0 S_line               // Set Loop Start and End addresses

        copya   p14,r2               // saves p for pointer manipulations on the first
                                     // matrix (line and column loop)
        SETLE0  (E_line-16)


        g1?     sub r2,r2,1      // start of the computation for the new value r2
        SETULS1 S_column

        shru    r2,r2,1          // using both DU with our short data during MAC
                                 // operations, corresponding loop counter can be
                                 // divided by 2
        SETLE1  (E_column-16)
```

```
        sub     r2,r2,1          // loop 2= coef loop executed (p/2)-1 if p is even,
                                 // ((p-1)/2)-1) if p is odd
        copyc   LC0,r0           // loop 0 = line loop is executed n times


        copya   p5,r1            // saves m in p5 for pointer manipulations
                                 // on the second matrix (coef loop)
        copyc   LC2,r2           // LC2 loaded with the new computed value r2

        sub     r4,r4,1
        copyc   LC1,r1           // loop 1 = column loop is executed m times

        fbpos   g2,r2,0          // g2=~g1
        SETULS2 S_coef

        copya   p4,r4            // saves N in p4
        SETLE2  (E_coef-16)

       .presliw
        sliwmd


S_line:
S_column:    ldlh   r14,@(p2!+p5) // short data of the second matrix can only be
                                  // loaded one a time because addresses of
                                  // both useful data are not consecutive
             ldw    r13,@(p1!+4)  // one step is executed outside the loop 3
                                  // to skip the re initialization of r12 and r15

             mpssll r12,r13,r14   // multiplies-accumulates the lower half-word
                                  // in r12 if j1 is odd
             nop


             ldhh   r14,@(p2!+p5) // puts both useful short data in r14
             nop
             nop
             mpsshh r15,r13,r14   // multiplies the upper half-words in r15


        g1?  ldlh   r13,@(p1?+2)  // loads separately the following half-word coeffi-
cients

        g1?  ldlh   r14,@(p2?+p5)    // if p is odd. After this, coefficients could be
                                     // loaded and computed word by word.
        g1?  massll r12,r12,r13,r14  // computes the multiplication-accumulation in r12
             nop



S_coef:      ldlh   r14,@(p2!+p5)
             ldw    r13,@(p1!+4)     // Loads corresponding data in r13 and r14
                                     // for the MAC operation
             massll r12,r12,r13,r14  // multiplies-accumulates (32 bits)in r12
                                     // with the lower half-words
             nop

             ldhh   r14,@(p2!+p5)
             nop
             nop
             masshh r15,r15,r13,r14  // multiplies-accumulates (32 bits) in r15
                                     // with the upper half-words
```

```
E_coef:          add     r12,r12,r15      // adds the upper and lower half-words to get
                                          // the right coefficient
                 nop
                 subha   p2,p2,p4         // p2<- start address of the following column
                 subha   p1,p1,p14        // p1<- start address of the corresponding
line



                 nop
                 nop
                 nop
                 sdh     @(p0!+2),r12     // stores the final result



E_column:
                 nop
                 nop
                 subha   p2,p2,p5         // p2<- start address of the second matrix
                 addha   p1,p1,p14        // p1<- address of the following line


E_line:          nop
                 nop
                 nop
                 gp32md


//  code ends here

         poprts  r4-r5,p4-p5,lr1,lr2
         .type   mMul_16_16_16,@function
         .size   mMul_16_16_16,.-mMul_16_16_16
         .globl  __callee.mMul_16_16_16.v.pppiii
```

4.24 - mMul_16_16_32

```
/*-------------------------------------------------------------
* FUNCTION : mMul_16_16_32 .
* Project component : ST100 GENLIB
* File Name : mMul_16_16_32.s
*  Purpose : Matrix multiply 16-bit integer, result on 32-bit integer (mat1 * mat2
->matRes)
* GP32 instruction set
*-------------------------------------------------------------
*/

        .text
        .entry32
        .globl          mMul_16_16_32

.align 8
/*-------------------------------------------------------------
* Function name: void mMul_16_16_32 (int32 *matres, int16 *mat1,int16 *mat2, int16
row1,int16 col2,int16 col1)
*
* Purpose: Matrix multiply 16-bit integer, result on 32-bit integer (mat1 * mat2
->matRes)
* In:
*      - mat1 address of the matrix1
*      - mat2 address of the matrix2
*      - row1 number of rows of matrix1
*      - col2 number of columns of matrix2
*      - col1 number of columns of matrix1
* Out:
*      - matres address of result matrix
* In/Out:
*      -
* Read global variables
*      -
* Modified global variables
*      -
* Returns:
*      -
*-------------------------------------------------------------
*/
mMul_16_16_32:

        push        r4-r5,p4-p5,lr1,lr2 // saves used registers
        mpuull      r4,r1,r2            // prepares N=(pm-1) used for pointer manipula-
tions
                                        // on the second matrix (column loop)

        tbpos       g1,r2,0             // test of the parity of p to prepare word load-
ing
                                        // g1 =1 if p is odd
        SETULS0     S_line              // Set Loop Start and End addresses

        copya       p14,r2              // saves p for pointer manipulations on the
first
                                        // matrix (line and column loop)
        SETLE0      (E_line-16)


        g1?         sub r2,r2,1         // start of the computation for the new value r2
        SETULS1     S_column
```

```
        shru        r2,r2,1        // using both DU with our short data during MAC
                                   // operations, corresponding loop counter can be
                                   // divided by 2
        SETLE1      (E_column-16)

        sub         r2,r2,1        // loop 2= coef loop executed (p/2)-1 if p is even,
                                   // ((p-1)/2)-1) if p is odd
        copyc       LC0,r0         // loop 0 = line loop is executed n times


        copya       p5,r1          // saves m in p5 for pointer manipulations
                                   // on the second matrix (coef loop)
        copyc       LC2,r2         // LC2 loaded with the new computed value r2

        sub         r4,r4,1
        copyc       LC1,r1         // loop 1 = column loop is executed m times

        fbpos       g2,r2,0        // g2=~g1
        SETULS2     S_coef

        copya       p4,r4          // saves N in p4
        SETLE2      (E_coef-16)

       .presliw
        sliwmd


S_line:
S_column:       ldlh   r14,@(p2!+p5) // short data of the second matrix can only be
                                     // loaded one a time because addresses of
                                     // both useful data are not consecutive
                ldw    r13,@(p1!+4)  // one step is executed outside the loop 3
                                     // to skip the re initialization of r12 and r15

                mpssll r12,r13,r14   // multiplies-accumulates the lower half-word
                                     // in r12 if j1 is odd
                nop


                ldhh    r14,@(p2!+p5)    // puts both useful short data in r14
                nop
                nop
                mpsshh  r15,r13,r14      // multiplies the upper half-words in r15


        g1?     ldlh    r13,@(p1?+2)     // loads separately the following half-word
coefficients

        g1?     ldlh    r14,@(p2?+p5)    // if p is odd. After this, coefficients could
be
                                         // loaded and computed word by word.
        g1?     massll  r12,r12,r13,r14  // computes the multiplication-accumulation in
r12
                nop



S_coef:         ldlh    r14,@(p2!+p5)
                ldw     r13,@(p1!+4)     // Loads corresponding data in r13 and r14
                                         // for the MAC operation
                massll  r12,r12,r13,r14  // multiplies-accumulates (32 bits) in r12
                                         // with the lower half-words
                nop
```

```
                ldhh    r14,@(p2!+p5)
                nop
                nop
                masshh   r15,r15,r13,r14  // multiplies-accumulates (32 bits) in r15
                                          // with the upper half-words


E_coef:         add     r12,r12,r15       // adds the upper and lower half-words to get
                                          // the right coefficient
                nop
                subha   p2,p2,p4          // p2<- start address of the following column
                subha   p1,p1,p14         // p1<- start address of the corresponding
line




                nop
                nop
                nop
                sdw     @(p0!+4),r12      // stores the final result



E_column:
                nop
                nop
                subha   p2,p2,p5          // p2<- start address of the second matrix
                addha   p1,p1,p14         // p1<- address of the following line



E_line:         nop
                nop
                nop
                gp32md


// my code ends here

        poprts  r4-r5,p4-p5,lr1,lr2
        .type   mMul_16_16_32,@function
        .size   mMul_16_16_32,.-mMul_16_16_32
        .globl  __callee.mMul_16_16_32.v.pppiii
```

4.25 - mMul_q15_q15_q15

```
/*-------------------------------------------------------------
* FUNCTION : mMul_q15_q15_q15
* Project component : ST100 GENLIB
* File Name : mMul_q15_q15_q15.s
* Purpose : Matrix multiply 16-bit fractional, result on 16-bit fractional.
* GP32 instruction set
*-------------------------------------------------------------
*/

        .text
        .entry32
        .globl          mMul_q15_q15_q15

.align 8
/*-------------------------------------------------------------
* Function name: void mMul_q15_q15_q15 (fract16 *matres, fract16 *mat1, fract16 *mat2,
int16 row1,int16 col2,int16 col1)
*
* Purpose: Matrix multiply 16-bit fractional, result on 16-bit fractional.
* In:
*      - mat1 address of the matrix1
*      - mat2 address of the matrix2
*      - row1 number of rows of matrix1
*      - col2 number of columns of matrix2
*      - col1 number of columns of matrix1
* Out:
*      - matres address of result matrix
* In/Out:
*      -
* Read global variables
*      -
* Modified global variables
*      -
* Returns:
*      -
*-------------------------------------------------------------
*/
mMul_q15_q15_q15:

        push    r4-r5,p4-p5,lr1,lr2 // saves used registers
        mpuull  r4,r1,r2            // prepares N=(pm-1) used for pointer manipulations
                                    // on the second matrix (column loop)

        tbpos   g1,r2,0          // test of the parity of p to prepare word loading
                                 // g1 =1 if p is odd
        SETULS0 S_line           // Set Loop Start and End addresses

        copya   p14,r2           // saves p for pointer manipulations on the first
                                 // matrix (line and column loop)
        SETLE0  (E_line-16)


        g1?     sub r2,r2,1      // start of the computation for the new value r2
        SETULS1 S_column

        shru    r2,r2,1          // using both DU with our short data during MAC
                                 // operations, corresponding loop counter can be
                                 // divided by 2
        SETLE1  (E_column-16)
```

```
        sub     r2,r2,1          // loop 2= coef loop executed (p/2)-1 if p is even,
                                 // ((p-1)/2)-1) if p is odd
        copyc   LC0,r0           // loop 0 = line loop is executed n times


        copya   p5,r1            // saves m in p5 for pointer manipulations
                                 // on the second matrix (coef loop)
        copyc   LC2,r2           // LC2 loaded with the new computed value r2

        sub     r4,r4,1
        copyc   LC1,r1           // loop 1 = column loop is executed m times

        fbpos   g2,r2,0          // g2=~g1
        SETULS2 S_coef

        copya   p4,r4            // saves N in p4
        SETLE2  (E_coef-16)

       .presliw
        sliwmd


S_line:
S_column:    ldlh   r14,@(p2!+p5) // short data of the second matrix can only be
                                  // loaded one a time because addresses of
                                  // both useful data are not consecutive
             ldw    r13,@(p1!+4)  // one step is executed outside the loop 3
                                  // to skip the re initialization of r12 and r15

             mpfll  r12,r13,r14   // multiplies-accumulates the lower half-word
                                  // in r12 if j1 is odd
             nop


             ldhh   r14,@(p2!+p5) // puts both useful short data in r14
             nop
             nop
              mpfhh r15,r13,r14   // multiplies the upper half-words in r15


        g1?    ldlh   r13,@(p1?+2)      // loads separately the following half-word
coefficients

        g1?    ldlh   r14,@(p2?+p5)     // if p is odd. After this, coefficients could
be
                                        // loaded and computed word by word.
        g1?    mafll  r12,r12,r13,r14   // computes the multiplication-accumulation in
r12
               nop


S_coef:       ldlh  r14,@(p2!+p5)
              ldw   r13,@(p1!+4)        // Loads corresponding data in r13 and r14
                                        // for the MAC operation
              mafll r12,r12,r13,r14     // multiplies-accumulates (32 bits)in r12
                                        // with the lower half-words
              nop

              ldhh  r14,@(p2!+p5)
              nop
              nop
              mafhh r15,r15,r13,r14     // multiplies-accumulates (32 bits) in r15
                                        // with the upper half-words
```

```
E_coef:          add     r12,r12,r15      // adds the upper and lower half-words to get
                                          // the right coefficient
                 nop
                 subha   p2,p2,p4         // p2<- start address of the following column
                 subha   p1,p1,p14        // p1<- start address of the corresponding
line



                 nop
                 shr     r12,r12,16
                 nop
                 sdh     @(p0!+2),r12     // stores the final result



E_column:
                 nop
                 nop
                 subha   p2,p2,p5         // p2<- start address of the second matrix
                 addha   p1,p1,p14        // p1<- address of the following line


E_line:          nop
                 nop
                 nop
                 gp32md


// my code ends here

         poprts  r4-r5,p4-p5,lr1,lr2
         .type   mMul_q15_q15_q15,@function
         .size   mMul_q15_q15_q15,.-mMul_q15_q15_q15
         .globl  __callee.mMul_q15_q15_q15.v.pppiii
```

4.26 - mMul_q15_q15_q31

```
/*-------------------------------------------------------------
* FUNCTION : mMul_q15_q15_q31
* Project component : ST100 GENLIB
* File Name : mMul_q15_q15_q31.s
* Purpose : Matrix multiply fractional 16-bit, result on fractional 32-bit.
* GP32 instruction set
*-------------------------------------------------------------
*/

        .text
        .entry32
        .globl          mMul_q15_q15_q31

.align 8
/*-------------------------------------------------------------
* Function name: void mMul_q15_q15_q31 (fract32 *matres, fract16 *mat1,fract16 *mat2,
int16 row1,int16 col2,int16 col1)
*
* Purpose: Matrix multiply fractional 16-bit, result on fractional 32-bit.
* In:
*      - mat1 address of the matrix1
*      - mat2 address of the matrix2
*      - row1 number of rows of matrix1
*      - col2 number of columns of matrix2
*      - col1 number of columns of matrix1
* Out:
*      - matres address of result matrix
* In/Out:
*      -
* Read global variables
*      -
* Modified global variables
*      -
* Returns:
*      -
*-------------------------------------------------------------
*/
mMul_q15_q15_q31:

        push    r4-r5,p4-p5,lr1,lr2 // saves used registers
        mpuull  r4,r1,r2            // prepares N = (pm-1) used for pointer manipulations
                                    // on the second matrix (column loop)

        tbpos   g1,r2,0             // test of the parity of p to prepare word loading
                                    // g1 =1 if p is odd
        SETULS0 S_line              // Set Loop Start and End addresses

        copya   p14,r2              // saves p for pointer manipulations on the first
                                    // matrix (line and column loop)
        SETLE0  (E_line-16)


        g1?     sub r2,r2,1         // start of the computation for the new value r2
        SETULS1 S_column

        shru    r2,r2,1             // using both DU with our short data during MAC
                                    // operations, corresponding loop counter can be
                                    // divided by 2
        SETLE1  (E_column-16)
```

```
        sub     r2,r2,1         // loop 2= coef loop executed (p/2)-1 if p is even,
                                // ((p-1)/2)-1) if p is odd
        copyc   LC0,r0          // loop 0 = line loop is executed n times


        copya   p5,r1           // saves m in p5 for pointer manipulations
                                // on the second matrix (coef loop)
        copyc   LC2,r2          // LC2 loaded with the new computed value r2

        sub     r4,r4,1
        copyc   LC1,r1          // loop 1 = column loop is executed m times

        fbpos   g2,r2,0         // g2=~g1
        SETULS2 S_coef

        copya   p4,r4           // saves N in p4
        SETLE2  (E_coef-16)

       .presliw
        sliwmd


S_line:
S_column:   ldlh   r14,@(p2!+p5)    // short data of the second matrix can only be
                                    // loaded one a time because addresses of
                                    // both useful data are not consecutive
            ldw    r13,@(p1!+4)     // one step is executed outside the loop 3
                                    // to skip the re initialization of r12 and r15

            mpfll  r12,r13,r14      // multiplies-accumulates the lower half-word
                                    // in r12 if j1 is odd
            nop


            ldhh   r14,@(p2!+p5)    // puts both useful short data in r14
            nop
            nop
            mpfhh  r15,r13,r14      // multiplies the upper half-words in r15


        g1?   ldlh    r13,@(p1?+2)    // loads separately the following half-word coefficients

        g1?   ldlh    r14,@(p2?+p5)  // if p is odd. After this, coefficients could be
                                     // loaded and computed word by word.
        g1?   mafll   r12,r12,r13,r14// computes the multiplication-accumulation in r12
              nop


S_coef:       ldlh    r14,@(p2!+p5)
              ldw     r13,@(p1!+4)      // Loads corresponding data in r13 and r14
                                        // for the MAC operation
              mafll   r12,r12,r13,r14   // multiplies-accumulates (32 bits)in r12
                                        // with the lower half-words
              nop

              ldhh    r14,@(p2!+p5)
              nop
              nop
              mafhh   r15,r15,r13,r14   // multiplies-accumulates (32 bits) in r15
                                        // with the upper half-words
```

```
E_coef:          add     r12,r12,r15      // adds the upper and lower half-words to get
                                          // the right coefficient
                 nop
                 subha   p2,p2,p4         // p2<- start address of the following column
                 subha   p1,p1,p14        // p1<- start address of the corresponding
line

                 nop
                 nop
                 nop
                 sdw     @(p0!+4),r12     // stores the final result

E_column:
                 nop
                 nop
                 subha   p2,p2,p5         // p2<- start address of the second matrix
                 addha   p1,p1,p14        // p1<- address of the following line

E_line:          nop
                 nop
                 nop
                 gp32md

// my code ends here

         poprts  r4-r5,p4-p5,lr1,lr2
         .type   mMul_q15_q15_q31,@function
         .size   mMul_q15_q15_q31,.-mMul_q15_q15_q31
         .globl  __callee.mMul_q15_q15_q31.v.pppiii
```

4.27 - mVM_8_16_16

```
/*-------------------------------------------------------------
* FUNCTION : mVM_8_16_16.
* Project component : ST100 GENLIB
* File Name : mVM_8_16_16.s
*  Purpose  :  Matrix  vector  multiply  (mat(8  bits  integer)  *  vec(16-bit  integer)
->vecRes(16-bit integer))
*-------------------------------------------------------------
*/


        .text
        .entry32
        .globl          mVM_8_16_16

.align 8
/*-------------------------------------------------------------
* Function name: void mVM_8_16_16 (int16 *vecRes, char *mat, int16 *vec, int16 matR,int16
matC)

* Purpose:  Matrix vector multiply (mat(8 bits integer) * vec(16-bit integer)
->vecRes(16-bit integer))
* In:
*      - vecRes address of the result vector
*      - mat address of the matrix
*      - vec address of the vector
*      - matR number of rows of the matrix
*      - matC number of columns of matrix
* Out:
*      - vecRes point on the result vector
* In/Out:
*      -
* Read global variables
*      -
* Modified global variables
*      -
* Returns:
*      -
*-------------------------------------------------------------
*/
mVM_8_16_16:
          push    LR1
          setuls0 S_column

          copyc   LC0,r0
          setuls1 S_coef

          copya   p14,r1

          shr     r1,r1,1
          sub     r1,r1,1
          setle0  (E_column-4)
          copyc   LC1,r1

          setle1  (E_coef-4)

          .presliw
          sliwmd

S_column: ldbp    r0,@(p1!+2)
          ldw     r1,@(p2!+4)
```

```
            mpssll  r12,r0,r1
            mpsshh  r13,r0,r1

S_coef:     ldbp    r0,@(p1!+2)
            ldw     r1,@(p2!+4)
            massll  r12,r12,r0,r1
            masshh  r13,r13,r0,r1

E_coef:
            addcw   r14,r12,r13
            nop
            subha   p2,p2,p14
            nop

            shlcw   r2,r14,16
            nop
            nop
            sdf     @(p0!+2),r2


E_column:   nop
            nop
            nop
            gp32md

        poprts  LR1
        .type   mVM_8_16_16,@function
        .size   mVM_8_16_16,.-mVM_8_16_16
        .globl  __callee.mVM_8_16_16.v.pppii
```

4.28 - mVM_q7_q15_q15

```
/*-------------------------------------------------------------
* FUNCTION : mVM_q7_q15_q15
* Project component : ST100 GENLIB
* File Name : mVM_q7_q15_q15.s
* Purpose : Matrix vector multiply (mat(8 bits fractional) * vec(16-bit fractional)
->vecRes(16-bit fractional)).
*-------------------------------------------------------------
*/

       .text
       .entry32
       .globl              mVM_q7_q15_q15

.align 8
/*-------------------------------------------------------------
* Function name: void mVM_q7_q15_q15 (fract16 *vecRes, fract8 *mat, fract16 *vec, int16
matR,int16 matC)
*
* Purpose: Matrix vector multiply (mat(8 bits fractional) * vec(16-bit fractional)
->vecRes(16-bit fractional)).
* In:
*      - vecRes address of the result vector
*      - mat address of the matrix
*      - vec address of the vector
*      - matR number of rows of the matrix
*      - matC number of columns of matrix
* Out:
*      - vecRes point on the result vector
* In/Out:
*      -
* Read global variables
*      -
* Modified global variables
*      -
* Returns:
*      -
*-------------------------------------------------------------
*/
mVM_q7_q15_q15:
       push         LR1
       setuls0      S_column

       copyc        LC0,r0
       setuls1      S_coef

       copya        p14,r1

       shr          r1,r1,1
       sub          r1,r1,1
       setle0       (E_column-4)
       copyc        LC1,r1

       setle1       (E_coef-4)

       .presliw
       sliwmd

S_column:           ldbp   r0,@(p1!+2)
                    ldw    r1,@(p2!+4)
                    mpfll  r12,r0,r1
```

```
                    mpfhh  r13,r0,r1

S_coef:             ldbp   r0,@(p1!+2)
                    ldw    r1,@(p2!+4)
                    mafll  r12,r12,r0,r1
                    mafhh  r13,r13,r0,r1

E_coef:
                    addcw  r14,r12,r13
                    nop
                    subha  p2,p2,p14
                    nop

                    shlcw  r2,r14,8
                    nop
                    nop
                    sdf    @(p0!+2),r2


E_column:           nop
                    nop
                    nop
                    gp32md


       poprts       LR1
       .type        mVM_q7_q15_q15,@function
       .size        mVM_q7_q15_q15,.-mVM_q7_q15_q15
       .globl       __callee.mVM_q7_q15_q15.v.pppii
```

4.29 - q31_energy_q15

```
/*-------------------------------------------------------------
* FUNCTION : q31_energy_q15
* Project component : ST100 GENLIB
* File Name : q31_energy_q15.s
* Purpose : Sum of square of a 16-bit fractional vector. Result is 32-bit fractional.
*-------------------------------------------------------------
*/

       .text
       .entry32
       .globl              q31_energy_q15

/*-------------------------------------------------------------
* Function name: fract32 q31_energy_q15 (fract16 *vec, int16 size)
*:
* Purpose: Sum of square of a 16-bit fractional vector. Result is 32-bit fractional.
*
* In:
*      - vec address of vector
*      - size number of element in the vector (multiple of 2)
* Out:
*      -
* In/Out:
*      -
* Read global variables
*      -
* Modified global variables
*      -
* Returns:
*      - energy
*-------------------------------------------------------------
*/

q31_energy_q15:

       sub           r0,r0,+2
       setuls0       energy_start

       shrw          r0,r0,+1
       setle0        energy_end-16

       copyc         c2,r0
       .presliw
       SLIWmd

       ldh           r13,@(p0+2)
       ldh           r2,@(p0?+4)
       mpfcll        r12,r13,r13
       mpfcll        r3,r2,r2

energy_start:

       ldh           r13,@(p0+2)
       ldh           r2,@(p0?+4)
       mafcll        r3,r3,r2,r2
       mafcll        r12,r12,r13,r13

energy_end:

       nop
```

```
addcw         r0,r3,r12
nop
GP32md

rts

.type         q31_energy_q15,@function
.size         q31_energy_q15,.-q31_energy_q15
.globl        __callee.q31_energy_q15.i.pi
```

4.30 - q31_vMul_q15

```
/*-------------------------------------------------------------
* FUNCTION : q31_vMul_q15
* Project component : ST100 GENLIB
* File Name : q31_vMul_q15.s
* Purpose : Vector multiply (vec1(16-bit fractional) x vec2(16-bit fractional)
->res(32-bit fractional)).
* GP32 instruction set
*-------------------------------------------------------------
*/

       .text
       .entry32
       .globl q31_vMul_q15


/*-------------------------------------------------------------
* Function name : fract32 q31_vMul_q15 (fract16 *vec1,fract16 *vec2, int16 size)
* Purpose :Vector multiply (vec1(16-bit fractional) x vec2(16-bit fractional)
->res(32-bit fractional))
*
* In :
*      - vec1 address of vector 1
*      - vec2 address of vector 2
*      - size number of element in the vectors
* Out:
*      -
* In/Out:
*      -
* Read global variables
*      -
* Modified global variables
*      -
* Returns:
*      - result
*-------------------------------------------------------------
*/

q31_vMul_q15:

       sub r0,r0,2
       shr r0,r0,1
       copyc lc0,r0

       SETULS0      _vdot_loop_start    // LSTART0 vdot_loop_start
       SETLE0       _vdot_loop_end - 16 // LVLIWE0 vdot_loop_end

       .presliw                         // sliwmd
       sliwmd

       //G0-sliw
       LDW          R1, @(P1 !+ 4)      // R1 s= [P1  !+ 4]
       LDW          R2, @(p0 !+ 4)      // R2 s= [p0  !+ 4]
       MPFCLL       R3, R1, R2          // R3 = R1L * R2L
       MPFCHH       R0, R1, R2          // R4 = R1H * R2H

_vdot_loop_start:

       //G0-sliw
       LDW          R1, @(P1 !+ 4)      // R1 s= [P1  !+ 4]
       LDW          R2, @(p0 !+ 4)      // R2 s= [p0  !+ 4]
```

```
        MAFCLL          R3, R3, R1, R2        // R3 = R3 + R1L * R2L
        MAFCHH          R0, R0, R1, R2        // R4 = R4 + R1H * R2H

_vdot_loop_end:

        //G1-sliw
        NOP
        ADD             R0, R3, R0            // R3 = R3 + R4
        NOP
        GP32md                                // gp32md


        poprts

        .type           q31_vMul_q15,@function
        .size           q31_vMul_q15, .-q31_vMul_q15
        .globl          __callee.q31_vMul_q15.i.ppi
```

4.31 - vAdd_16

```
/*---------------------------------------------------------------
* FUNCTION : vAdd_16
* Project component : ST100 GENLIB
* File Name : vAdd_16.s
* Purpose : Vector addition (integer or fractional): vec1(16-bit) + vec2(16-bit)
->vecRes(16-bit)
* GP32 instruction set
*---------------------------------------------------------------
*/



       .text
       .entry32
       .globl vAdd_16

/*---------------------------------------------------------------
* Function name: void vAdd_16 (int16 *vecRes, int16 *vec1,int16 *vec2, int16 size)
*
* Purpose:Vector addition (integer or fractional): vec1(16-bit) + vec2(16-bit)
->vecRes(16-bit)
* In:
*      - vec1 address of vector 1
*      - vec2 address of vector 2
*      - size number of element of vectors
* Out:
*      - vecRes address of result vector
* In/Out :
*      -
* Read global variables
*      -
* Modified global variables
*      -
* Returns:
*      -
*---------------------------------------------------------------
*/

vAdd_16:
/*    DO_VEC_SUM:    Initialization          */

       she           r0,r0,1
       copy          LC0,r0

       SETLE0        _fattened - 4             // LGP32E0 _fattened
       SETULS0       _vadd_start               // LSTART0 _vadd_start
       nop
       nop
       nop
/*    VADD:    Kernel                         */
.align 16
_vadd_start:
       LDP           R0, @(P1 !+ 4)            // R0 s= [P1  !+ 4]
       LDP           R1, @(P2 !+ 4)            // R1 s= [P2  !+ 4]

       ADDP          R2, R0, R1                // R2 p= R0 + R1
       SDP           @(P0 !+ 4), R2            // [P0!+ 4] = R2
.align 8
_vadd_end:
```

```
/*    DO_VEC_SUM:    Termination                  */

       poprts
       .type         vAdd_16,@function
       .size         vAdd_16,.-vAdd_16
       .globl        __callee.vAdd_16.v.pppi
```

5 - REFERENCES

[1] ST120 DSP - MCU Core Reference Guide.
[2] ST120 DSP - MCU Programming manual.

littlediode
COMPONENTS FOR ALL

Littlediode supplies new, hard to find or obsolete electronic components and semiconductors all over the world.

With over two million different components listed you are sure to find the part you need.

Feel free to visit us today at our online store:

LittleDiode.com

Looking forward to providing you with the best possible service.