Reference Guide Low-Energy Accelerator (LEA) Commands

TEXAS INSTRUMENTS

ABSTRACT

This reference manual describes the behavior of the commands implemented in the LEA Code ROM and gives a short overview of the LEA hardware registers and their functions. The commands are explained in a brief mathematical notation. Programming examples are based on native coding and native data elements.

This manual addresses programmers that use fast native-style coding. For writing elegant code, see the Digital Signal Processing (DSP) Library for MSP430 Microcontrollers.

Table of Contents

1 LEA Operation	3
1.1 Initializing and Enabling the LEA	
1.2 Invoking a Command	
1.2.1 Command Invoke Methods	
1.2.2 Command Execution in Low-Power Modes	
1.3 LEADONE and LEAFREE Indications	
1.3.1 LEA Done Indication	7
1.3.2 LEAFREE Indication	
2 LEA Commands	<mark>8</mark>
2.1 Group 1: Basic Pointwise Vector or Matrix Operations	8
2.1.1 LEACMD_ADDLONGMATRIX (rADDMI)	9
2.1.2 LEACMDADDMATRIX (rADDMs)	
2.1.3 LEACMDMPYMATRIX (rMPYMsf)	.11
2.1.4 LEACMDMPYLONGMATRIX (rMPYMIf)	
2.1.5 LEACMDMPYCOMPLEXMATRIX (cMPYMsf)	
2.1.6 LEACMDSUBMATRIX (rSUBMs)	
2.1.7 LEACMDSUBLONGMATRIX (rSUBI)	
2.2 Group 2: Basic Vector MAC Operations	
2.2.1 LEACMDMAC (rMACsf)	
2.2.2 LEACMDMAC3 (rMAC3sf)	
2.2.3 LEACMDSCALEDMAC (sMACsf)	
2.3 Group 3: MAC, Point-Wise FIR, Correlation, and Convolution Operations	
2.3.1 LEACMDMACMATRIX (rMACMsf)	
2.3.2 LEACMDMACLONGMATRIX (rMACMIf)	
2.3.3 LEACMDMACCOMPLEXMATRIX (cMACMsf)	
2.3.4 LEACMDMACCOMPLEXCONJUGATEMATRIX (cMACMCONJsf)	
2.4 Group 4: Basic Min/Max Vector Search on 16B Data	
2.4.1 LEACMDMAX (rMAXs)	
2.4.2 LEACMDMIN (rMINs)	
2.4.3 LEACMDMAXUNSIGNED (rMAXus)	
2.4.4 LEACMDMINUNSIGNED (rMINus)	
2.5 Group 5: Generic Min/Max Vector Search on 32B Data	
2.5.1 LEACMDMAXLONGMATRIX (rMAXMI)	
2.5.2 LEACMDMINLONGMATRIX (rMINMI)	
2.5.3 LEACMDMAXUNSIGNEDLONGMATRIX (rMAXMul)	
2.5.4 LEACMDMINUNSIGNEDLONGMATRIX (rMINMul)	
2.6 Group 6: Basic Min/Max Vector Search on Dual 16B Data	
2.6.1 LEACMD_MAXMATRIX (rMAXMs)	
2.6.2 LEACMDMINMATRIX (rMINMs)	
2.6.3 LEACMDMAXUNSIGNEDMATRIX (rMAXMus)	
2.6.4 LEACMD MINUNSIGNEDMATRIX (rMINMus)	.36

1



2.7 Group 7: Block FIR, Correlation, Convolution Functions	37
2.7.1 LEACMDFIR (rFIRsf)	
2.7.2 LEACMDFIRLONG (rFIRIf)	
2.7.3 LEACMD FIRCOMPLEX (cFIRsf)	
2.7.4 LEACMD FIRCOMPLEXLONG (cFIRIf)	
2.7.5 LEACMD SCALEDFIR (sFIRsf)	
2.8 Group 8: Taylor McLaurin Functions on Pointwise Vectors or Matrices	
2.8.1 LEACMD POLYNOMIAL (rPOLYsf)	
2.8.2 LEACMD POLYNOMIALLONG (rPOLYIf)	
2.9 Group 9: Fast Fourier Transforms and Inverses	
2.9.1 LEACMD FFTCOMPLEXAUTOSCALING (cFFTsf as)	
2.9.2 LEACMD FFTCOMPLEXFIXSCALING (cFFTsf fs)	
2.9.3 LEACMD FFTCOMPLEXLONG (cFFTIf)	
2.10 Group 10: Bit Reversed Pre-Sort Functions for FFTs	
2.10.1 LEACMD BITREVERSECOMPLEXEVEN (cBREVEsf),	
	<mark>56</mark>
2.10.2 LEACMD BITREVERSECOMPLEXLONGEVEN (cBREVEIf),	
LEACMDBITREVERSECOMPLEXLONGODD (cBREVOIf)	<mark>58</mark>
2.11 Group 11: Post Operation for Real FFTs	<mark>59</mark>
2.11.1 LEACMDFFTSPLIT (SPLIFsf/rFFTsf), LEACMDFFTLONGSPLIT (SPLITIf/rFFTIf)	59
2.12 Group 12: De-Interleave and Sort Functions	60
2.12.1 LEACMDDEINTERLEAVEEVENEVEN (rSPLITEEs),	
LEACMDDEINTERLEAVEEVENODD (rSPLITEOs), LEACMDDEINTERLEAVEODDEVEN (rSPLITOEs)	
LEACMDDEINTERLEAVEODDODD (rSPLITOOs), LEACMDDEINTERLEAVELONG (rSPLITI)	<mark>61</mark>
2.13 Group A: Programming Structure Functions	
2.13.1 LEACMDMOVELONGLIST (LSTMOVI)	
2.13.2 LEACMD_SUSPEND (SUSPEND)	
2.13.3 LEACMDRESUME (RESUME)	
2.14 Group B: Special Functions for Math and Signal Processing	
2.14.1 LEACMDMPYMATRIXROW	
2.14.2 LEACMDPOLYNOMIALSCALAR	
2.14.3 LEACMD_IIRBQ1 (rBQ1sf)	
2.14.4 LEACMD_IIRBQ2 (rBQ2sf)	
2.14.5 LEACMD_IIRBQ2EXTENDED (rBQ2Xsf)	
2.14.6 LEACMD WINDOW	
3 LEA Hardware	
3.1 LEA ROM Constants	
3.2 LEA Timer	
3.3 LEA and Debug	
3.4 LEA Hardware Registers	
4 Revision History	78

Trademarks

All trademarks are the property of their respective owners.

1 LEA Operation

The LEA provides a series of commands that perform mathematical operation on single samples, trains of samples, sample streams, point-wise vectors and matrices, and on scalar values. The commands are separated into groups based on their primary operations and parameter blocks. Some mathematical or DSP skills are required to immediately recognize what command suits the application best.

Convolution and correlation are often treated as separate functions. However, the LEA has one elementary command that does both, and this command also performs other operations. Because many LEA commands can perform multiple functions, you might not find all of the algorithms that you expect. The LEA has implemented only the native ones, the bare minimum, as described in the following examples.

Example 1: Convolution, Correlation, and FIR filter are based on the same algorithm. How the pointers change makes the difference.

While time runs forward only for us, a train of samples stored in LEA memory allows you to go forward and backward "in time".

Example 2: FFT (fast Fourier transform) and iFFT (inverse FFT) are based on the same algorithm. The iFFT just uses a negative index for the results.

Example 3: The LEA does not provide a "move vector" function. This function can be emulated with a "vector add" operation using a "zero vector" as the neutral element. Mirroring vector can be implemented by running the source and destination pointers in opposite directions, and so forth.

LEA commands use parameter blocks to pass the arguments from the caller. The LEA has fourteen different command groups (see Table 1-1). Each command group has a dedicated structure defined. Commands of the same group share the same type of parameter block. For the functions of group 1-12 common generic parameter blocks are defined. This allows using the same control memory again. Functions of Group A and B are the exceptions. The parameter blocks of those functions are treated separately.

Group	Use
Group 1	Basic point-wise vector or matrix operations
Group 2	Basic vector MAC operations (windowing, scaling, general)
Group 3	MAC, point-wise FIR, correlation, convolution
Group 4	Basic min/max vector search operations on 16-bit data
Group 5	Generic min/max search operations on 32-bit data
Group 6	Generic min/max search operations on dual 16-bit data or complex
Group 7	Block based FIR, correlation, convolution
Group 8	Taylor functions or operations on point-wise vectors or matrices
Group 9	FFT or iFFT bank filtering (DIT-type)
Group 10	Bit reversed carry propagated pre-sort for DIT-FFTs
Group 11	FFT post operation for real points
Group 12	Vector or matrix de-interleave or sort functions
Group A	Programming structure or rearrange functions
Group B	Special functions for math, matrix, and DSP

 Table 1-1. LEA Command Groups

All data elements that the LEA needs to access (inputs, outputs, and parameter blocks) must be in the memory section *leaRAM*. Dedicated macros are available to help with this location.

1.1 Initializing and Enabling the LEA

The LEA is initialized by setting the registers LEACNF0, LEACNF1 and LEACNF2.

<pre>LEACNF0=0; LEACNF1=0; LEACNF2=LeaSP;</pre>	//init LEA
LEAPMCTL =LEACMDEN;	//enable LEA



1.2 Invoking a Command

The LEA performs operations on data stored in *leaRAM*. Two arguments are passed to LEA in the registers LEAPMS0 and LEAPMS1. The command itself and its operation mode are passed in the LEAPMCB register. If more than two arguments are required, LEAPMS1 points to a parameter block that holds the additional arguments. Because most commands use more than two arguments, some examples are given below.

The following example shows a MAC operation step by step using the corresponding LEA commands. Other commands are used in a similar way.

Invoking a command is done in three steps:

- 1. Declare a parameter block and input and output variable for MAC in *leaRAM*. (A parameter block is an array of 16 bit values that holds the arguments require for the function. As control structure it holds pointers, increment and count values required for the command. This structure is not required anymore after the command is invoked. In many cases it is wise to keep the structure if the same or a different command is invoked with only slight changes of parameter).
- 2. Initialize the input variables and parameters according to the function. (This is done either by a true initialization or by an previous running operation. For example, the output vector of one LEA command can be the input vector of another command).
- 3. To invoke a command, write to LEAPMS0, LEAPMS1 and, LEAPMCB. In most cases only the lower 16-bits of the registers are used.

The following is a complete test program.

```
#include <msp430.h>
//macros to simplify LEA code (usually put in a header file)
#define LEA EPRG(x) PRAGMA(#x) //auxiliary definition for NewLEA
#define NewLEA VAR(var) LEA EPRG(RETAIN(var)) LEA EPRG(DATA SECTION(var,".leaRAM"))
#define Ladr(x) ((unsigned short)((unsigned long)(x) & 0xffff)>>2)
#define Q15(x) (x)<1.0? ((x)>-1.0? (x)*0x8000:0x8000):0x7FFF
#define LeaSP 0x3C00/4 //define Lea-Stack to top of LEA memory {32 bit oriented adr.}
NewLEA VAR(Xi); NewLEA VAR(Yi); NewLEA VAR(Zo); NewLEA VAR(P); //declare variables in leaRAM
signed short Xi[]={Q15(-0.5), Q15(0.3), Q15(-0.4), Q15(0.5)}; //init with some test data
signed short Yi[]={Q15(-1) , Q15(-1) , Q15(-1) , Q15(-1) }; //init with some test data
signed long Zo[]={0x00000000}; //init space for result
short P[]={0,0,4,0,0,0};
                                    //here pre initialized parameters with \ensuremath{\text{N=4}}
void main(void) {
  WDTCTL = WDTPW | WDTHOLD; // Stop watchdog timer
  LEACNF0=0; LEACNF1=0; LEACNF2=LeaSP; //init LEA
  LEAPMCTL | =LEACMDEN;
                                               //enable LEA
  P[0]=Ladr(&Xi); //set X in parameters
  P[3]=Ladr(&Yi); //ditto with Y using macro;
P[4]=Ladr(&Zo); //address of Z to last element of parameters
LEAPMSOL=P[0]; //put 1st parameter directly in register ~PM.
                     //put 1st parameter directly in register ~PMSOL; here adr of X[]
 LEAPMS1L=Ladr(&P[2]); //put address of base parameter in ~PMS1L
LEAPMCBL=LEACMD_MAC; //start/invoke command
  for(;;);}
                            //have a look yourself
```

The parameter block for the command LEACMD__MAC holds four short values for the base parameter and two additional short parameters for the extension.

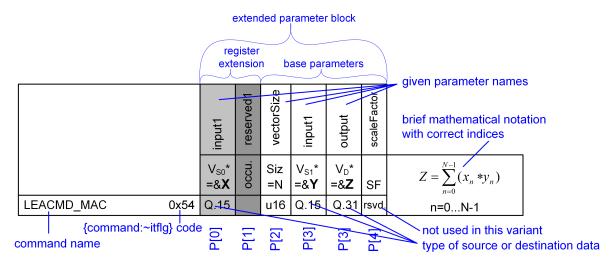


Figure 1-1. MAC Parameter Block

The values of the parameter structure P[] were set before the command is invoked. Invoking is done as described.

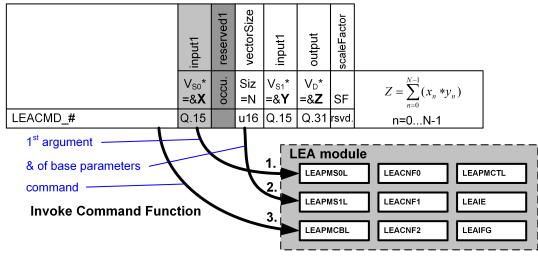


Figure 1-2. Invoking LEACMD__MAC

After the LEA completes operation, read the result memory location for the value 0.1 (0xCCD).

Note

Address boundaries, increment values, and counts are not checked for validity. Invalid values can corrupt the entire leaRAM content. Null pointers are a common source for errors (DSPlib checks for null pointers).

You can use *typedef* to simplify the handling of the parameter blocks. Whether the bare base parameter block or the extended version is used is up to the coding style. The following two examples show such struct definitions using native data types.



Exampe: MAC Extended Parameter Struct

typedef	struct {		
short	input1;	11	address of input1 vector containing values of short
short	reserved1;	11	reserved; may be used for context
short	vectorSize;	11	vector size */
short	input2;	11	address of input2 vector containing values of short
short	output;	11	address of output vector containing values of long
short	reserved2;	11	reserved, not used
}LEA n	nacParams;	11	extended parameter block of MAC

Example: MAC Base Parameter Struct

```
typedef struct {
   short vectorSize; // vector size */
   short input2; // address of input2 vector containing values of short
   short output; // address of output vector containing values of long
   short reserved2; // reserved, not used
   }LEA_macBPara; // base parameter block of MAC
```

1.2.1 Command Invoke Methods

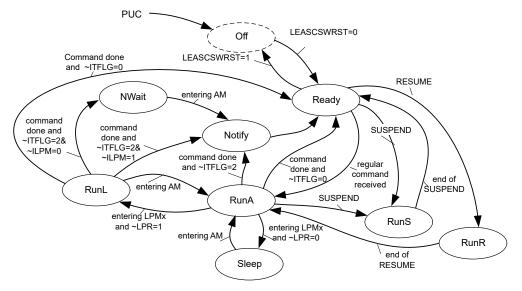
LEA Commands are invoked with LEAITFLG. Those interrupt and transaction flags define LEA's response on completion. LEAITFLG set to two and three will cause an interrupt upon completion. LEAFLG set to one and three will overwrite LEAPMDST without checking if the previous result has been picked up. The value of LEAITGLG does not influence LEADONE and LEAFREE indications. Table 1-2 gives an overview of the methods.

Table 1-2. LEA Command invoke Methods									
Invoke Method	LEA Command Response	LEAITFLG							
LEAPMCBL= <i>LEACMD_xxx</i> +0;	LEA command without any further indication	0							
LEAPMCBL= <i>LEACMD_xxx</i> +1;	LEA command with explicit result update	1							
LEAPMCBL= <i>LEACMD_xxx</i> +2;	LEA command with interrupt upon completion	2							
LEAPMCBL= <i>LEACMD_xxx</i> +3;	LEA command with interrupt and explicit result update	3							

Table 1-2. LEA Command Invoke Methods

1.2.2 Command Execution in Low-Power Modes

LEA provides two flags to define the operation behavior in low-power modes. LEALPR defines if a LEA command shall be continued when CPU enters LPM0 to LPM3. LEAILPM defines if a command completion interrupt shall be triggered while the CPU is in low-power mode LPM0 to LPM3.





1.3 LEADONE and LEAFREE Indications

LEADONE and LEAFREE are status flags used for software flow control.

1.3.1 LEA Done Indication

This flag indicates that an LEA operation has been completed, and a new result is available. The user may wait for this situation by software before picking up the result of the previous LEA operation. The done indication is reflected on LEACNF0.LEADONES and LEACNF1.LEADONEC.

- LEA Done behavior as seen in LEACNF0 and LEACNF1
- LEADONE is cleared after reset and RESTART (LEACNF0.SWRST=1).
- LEADONE is set on completion of a Command with LEAITFLG-field =1|3
- · LEADONE is cleared on read of LEAPMDST. (also on dummy reads).
- LEADONE is cleared after a SUSPEND operation is executed. The previous LEADONE status is being stored onto the LEA stack along with the suspended command context LEADONE is loaded from the previously saved LEA stack one a RESUME operation, thus restoring the saved core event context.

1.3.2 LEAFREE Indication

LEAFREE behavior as seen in LEACNF0 and LEACNF1:

- LEAFREE is set after reset and RESTART (LEACNF0.SWRST=1).
- LEAFREE is set after a operation with LEAITFLG =0|2 has been completed and its results have been written to LEA accessible memory or to the result register).
- LEAFREE is set as soon the result of a command with LEAITFLG=1|3 has been read from LEAPMDSTH register
- LEAFREE is set after a SUSPEND operation is executed. The previous LEAFREE status is being stored onto the LEA stack along with the suspended command context
- LEAFREE is cleared if a soon a command has been started.
- LEAFREE is loaded from the previously saved LEA stack one a RESUME operation, thus restoring the saved core event context

This flag indicates that LEA is free for new operations. The user may use wait for this situation by software before invoking a new command. The free indication is reflected on LEACNF0.LEAFREES and LEACNF1.LEAFREEC.

7

2 LEA Commands

The fourteen command groups of LEA are explained group by group. The data types IQ16, Q.15, IQ1.14, u16, s16, LEAadr are all of type short as native data type. IQ32, Q.31, IQ15.16, cQ.15 are of type long as native data type. LEA does not differentiate on the meaning and purpose given by programmers. It only understands 16 bit numbers and 32-bit numbers.

2.1 Group 1: Basic Pointwise Vector or Matrix Operations

Table 2-1 lists the parameter structure for this group.

Table 2-1. Group	1 Parameters:	Basic Pointwise	Vector or Matrix Operations
------------------	---------------	-----------------	-----------------------------

		input1	reserved1	vectorSize	input2	output	input10ffset	input2Offset	outputOffset	
		V _{S0} *	cu.	Siz	V _{S1} *	V _D *	V _{s0} +=	V _{S1} +=	V _D +=	
		=& X	8	=N	=&Y	=& Z	=X _i	=Y _i	=Z _i	
LEACMD_ADDMATRIX	0x40	IQ16		u16	IQ16	IQ16	s16	s16	s16	$z_n = x_n + y_n ;$
LEACMDADDLONGMATRIX	0x7C	IQ32		u16	IQ32	IQ32	s16	s16	s16	n = 0N-1
LEACMDMPYMATRIX	0x3C	Q.15		u16	Q.15	Q.15	s16	s16	s16	
LEACMDMPYLONGMATRIX	0x74	Q.31		u16	Q.31	Q.31	s16	s16	s16	$z_n = x_n * y_n ;$ n = 0N-1
LEACMDMPYCOMPLEXMATRIX	0x78	cQ.15		u16	cQ.15	cQ.15	s16	s16	s16	n = 0N-1
LEACMD_SUBMATRIX	0x70	IQ16		u16	IQ16	IQ16	s16	s16	s16	$z_n = x_n - y_n ;$
LEACMD_SUBLONGMATRIX	0xD4	IQ32		u16	IQ32	IQ32	s16	s16	s16	n = 0N-1

2.1.1 LEACMD_ADDLONGMATRIX (rADDMI)

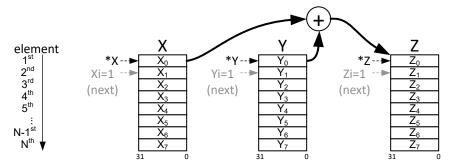
Vector X and Vector Y are point wise added and written to vector Z using a 32-bit signed addition. The results saturate to $0x7FFFFFEQ31(1-\epsilon)$ and $0x80000000\equiv Q31(-1)$.

Mathematical notation

 $z_n = x_n + y_n$ with n = 0 to (N-1)

Arguments

pointer to vector X as a 16-bit address value, pointing at an element of vector X pointer to vector Y as a 16-bit address value, pointing at an element of vector Y pointer to vector Z as a 16-bit address value, pointing at an element of vector Z increment step of Xi as a 16-bit signed value, *X is incremented with after each operation increment step of Yi as a 16-bit signed value, *Y is incremented with after each operation increment step of Zi as a 16-bit signed value, *Z is incremented with after each operation vector size N as a 16-bit unsigned value; defining the number of operations performed **Example** for Xi = Yi = Zi = 1, N = 8



Uses

- Pointwise addition of whole vectors of signed long and Q.31 type data
- Pointwise vector copy from X to Z , if Y is zero containing(0) vector as neutral element.
- Vector fill from X to Z, if Y is 0 and Xi=0, Yi=0, Zi=1. (for gapless fill)
- · Address mirrored additions and copy using negative Xi, Yi or Zi
- · Interleave or de-interleave and copy using other increment values.

- Alternative to an incrementing index order of operations a decrementing order is possible; then starting with pointers pointing to last elements.
- Matrices and complex numbers are treated as vectors.



2.1.2 LEACMD__ADDMATRIX (rADDMs)

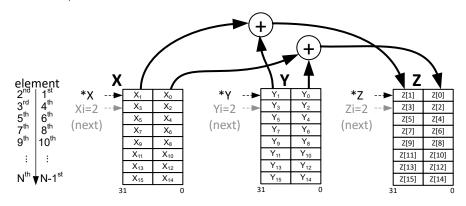
Vector X and Vector Y are point wise added and written to vector Z by adding two 16-bit signed values at once. The results saturate to $0x7FFF=Q15(1-\epsilon)$ and 0x8000=Q15(-1).

Mathematical notation

 $z_n = x_n + y_n$ with n = 0 to (N-1) {N: multiple of 2}

Arguments

pointer to vector X as a 16-bit address value, pointing at an element pair of vector X pointer to vector Y as a 16-bit address value, pointing at an element pair of vector Y pointer to vector Z as a 16-bit address value, pointing at an element pair of vector Z increment step of Xi as a 16-bit signed value, *X is incremented with {multiple of 2} increment step of Yi as a 16-bit signed value, *Y is incremented with {multiple of 2} increment step of Zi as a 16-bit signed value, *Z is incremented with {multiple of 2} vector size N as a 16-bit unsigned value; defining the number of operations performed **Example** for Xi = Yi = Zi = 2, N = 16



Uses

- Pointwise addition of whole vectors of signed short and Q.15 type data
- Pointwise vector copy from X to Z , if Y is zero containing(0) vector as
- Vector fill from X to Z, if Y is 0 and Xi=0, Yi=0, Zi=2. (for gapless fill)
- Upscaling of vectors (for example, X+X=2X)
- · Address mirrored additions and copy using negative Xi,Yi or Zi
- · Interleave or de-interleave and copy using other increment values

- Alternative to an incrementing index order of operations a decrementing order is possible; then starting with pointers pointing to last elements.
- Matrices and complex numbers are treated as vectors.
- · Pad odd vector lengths with zeroes to obtain predictable results
- · This command is also applied to SIMD vector pairs to save energy

2.1.3 LEACMD__MPYMATRIX (rMPYMsf)

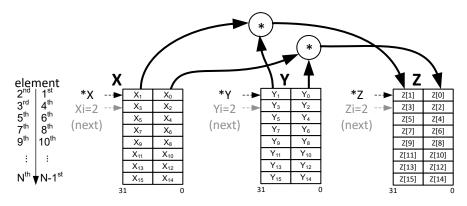
Vector X and Vector Y are point wise multiplied and written to vector Z by multiplying two 16-bit signed fractional values at once.

Mathematical notation

 $z_n = x_n \cdot y_n$ with n = 0 to (N-1) {N: multiple of 2}

Arguments

pointer to vector X as a 16-bit address value, pointing at an element pair of vector X pointer to vector Y as a 16-bit address value, pointing at an element pair of vector Y pointer to vector Z as a 16-bit address value, pointing at an element pair of vector Z increment step of Xi as a 16-bit signed value, *X is incremented with {multiples of 2} increment step of Yi as a 16-bit signed value, *Y is incremented with {multiples of 2} increment step of Zi as a 16-bit signed value, *Z is incremented with {multiples of 2} vector size N as a 16-bit unsigned value; defining the number of operations performed **Example** for Xi = Yi = Zi = 2, N = 16



Uses

- Pointwise multiplication of whole vectors of Q.15 type data
- Windowing function to smoothen start and end of vectors
- Downscaling of vectors

- alternative to an incrementing index order of operations a decrementing order is possible; then starting with pointers pointing to last elements.
- matrices and complex numbers are treated as vectors.
- · pad odd vector lengths with zeroes to obtain predictable results
- this command is also applied to SIMD vector pairs to save energy



2.1.4 LEACMD__MPYLONGMATRIX (rMPYMIf)

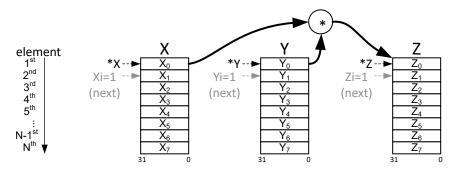
Vector X and Vector Y are point wise multiplied and written to vector Z using a 32-bit fractional multiplication.

Mathematical notation

 $z_n = x_n \times y_n$ with n = 0 to (N-1)

Arguments

pointer to vector X as a 16-bit address value, pointing at an element of vector X pointer to vector Y as a 16-bit address value, pointing at an element of vector Y pointer to vector Z as a 16-bit address value, pointing at an element of vector Z increment step of Xi as a 16-bit signed value, *X is incremented with after each operation increment step of Yi as a 16-bit signed value, *Y is incremented with after each operation increment step of Zi as a 16-bit signed value, *Z is incremented with after each operation vector size N as a 16-bit unsigned value; defining the number of operations performed **Example** for Xi = Yi = Zi = 1, N = 8



Uses

- Pointwise multiplication of whole vectors of Q.31 type data
- Windowing function to smoothen start and end of vectors
- Downscaling of vectors

- Alternative to an incrementing index order of operations a decrementing order is possible; then starting with
 pointers pointing to last elements.
- Matrices and complex numbers are treated as vectors.



2.1.5 LEACMD__MPYCOMPLEXMATRIX (cMPYMsf)

Vector X and Vector Y are point wise multiplied and written to vector Z using a 16-bit fractional complex multiplication.

The results saturate to $0x7FFF=Q15(1-\varepsilon)$ and 0x8000=Q15(-1).

Mathematical notation

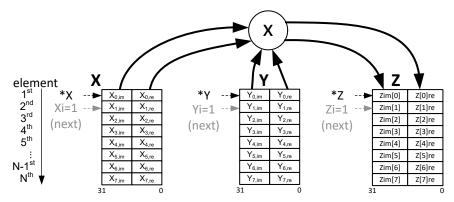
 $z_n = x_n \times y_n$ with n = 0 to (N-1), while $z_{n,re} = x_{n,re} \cdot y_{n,re} - x_{n,im} \cdot y_{n,im}$ and

 $z_{n,im} = x_{n,im} \cdot y_{n,re} + x_{n,re} \cdot y_{n,im}$

Arguments

pointer to vector X as a 16-bit address value, pointing at an element of vector X

pointer to vector Y as a 16-bit address value, pointing at an element of vector Y pointer to vector Z as a 16-bit address value, pointing at an element of vector Z increment step of Xi as a 16-bit signed value, *X is incremented with after each operation increment step of Yi as a 16-bit signed value, *Y is incremented with after each operation increment step of Zi as a 16-bit signed value, *Z is incremented with after each operation vector size N as a 16-bit unsigned value; defining the number of operations performed **Example** for Xi = Yi = Zi = 1, N = 8



Uses

- · Pointwise complex multiplication of whole vectors of complex Q.15 type data
- 2D rotate and mirror and stretch operation
- Vector CORDIC operations

- Alternative to an incrementing index order of operations a decrementing order is possible; then starting with pointers pointing to last elements.
- Complex numbers of higher order require SW support.



2.1.6 LEACMD__SUBMATRIX (rSUBMs)

Vector X and Vector Y are point wise subtracted and written to vector Z by subtracting two 16-bit signed values at once.

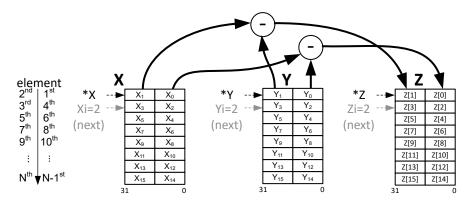
The results saturate to $0x7FFF=Q15(1-\epsilon)$ and 0x8000=Q15(-1).

Mathematical notation

 $z_n = x_n - y_n$ with n = 0 to (N-1) {N: multiple of 2}

Arguments

pointer to vector X as a 16-bit address value, pointing at an element pair of vector X pointer to vector Y as a 16-bit address value, pointing at an element pair of vector Y pointer to vector Z as a 16-bit address value, pointing at an element pair of vector Z increment step of Xi as a 16-bit signed value, *X is incremented with {multiple of 2} increment step of Yi as a 16-bit signed value, *Y is incremented with {multiple of 2} increment step of Zi as a 16-bit signed value, *Z is incremented with {multiple of 2} vector size N as a 16-bit unsigned value; defining the number of operations performed **Example** for Xi = Yi = Zi = 2, N = 16



Uses

- Pointwise addition of whole vectors of signed short and Q.15 type data
- Pointwise vector copy from X to Z , if Y is zero containing(0) vector as neutral element.
- Vector fill from X to Z, if Y is 0 and Xi=0, Yi=0, Zi=2. (for gapless fill)
- Address mirrored additions and copy using negative Xi, Yi or Zi
- · Interleave/de-interleave and copy using other increment values

- alternative to an incrementing index order of operations a decrementing order is possible; then starting with
 pointers pointing to last elements.
- matrices and complex numbers are treated as vectors.
- pad odd vector lengths with zeroes to obtain predictable results
- this command is also applied to SIMD vector pairs to save energy

2.1.7 LEACMD__SUBLONGMATRIX (rSUBI)

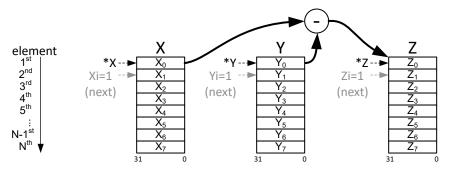
Vector X and Vector Y are point wise subtracted and written to vector Z using a 32-bit signed subtraction. The results saturate to $0x7FFFFFEQ31(1-\epsilon)$ and $0x80000000\equiv Q31(-1)$.

Mathematical notation

 $z_n = x_n - y_n$ with n = 0 to (N-1)

Arguments

pointer to vector X as a 16-bit address value, pointing at an element of vector Z pointer to vector Y as a 16-bit address value, pointing at an element of vector Y pointer to vector Z as a 16-bit address value, pointing at an element of vector Z increment step of Xi as a 16-bit signed value, *X is incremented with after each operation increment step of Yi as a 16-bit signed value, *Y is incremented with after each operation increment step of Zi as a 16-bit signed value, *Z is incremented with after each operation vector size N as a 16-bit unsigned value; defining the number of operations performed **Example** for Xi=Yi=Zi=1; N=8



Uses

- · Pointwise addition of whole vectors of signed long and Q.31 type data
- Pointwise vector copy from X to Z , if Y is zero containing(0) vector as neutral element.
- Vector fill from X to Z, if Y is 0 and Xi=0, Yi=0, Zi=1. (for gapless fill)
- · Address mirrored additions and copy using negative Xi, Yi, or Zi
- · Interleave/de-interleave and copy using other increment values

- Alternative to an incrementing index order of operations a decrementing order is possible; then starting with pointers pointing to last elements.
- Matrices and complex numbers are treated as vectors.

2.2 Group 2: Basic Vector MAC Operations

Table 2-2 lists the parameter structure for this group.

Table 2-2 G	roup 2 Parameters	s - Basic MAC Vector
	I up L I alameters	S - Dasic INAO VECIUI

		input1	reserved1	vectorSize	input2	output	scaleFactor		
		V _{s0} * =&X	occu.	Siz =N	V _{S1} * =& Y	V _D * =& Z	SF	N=2	on MAC3 $x_{n+1} \Leftrightarrow y_n \to z_0$ $x_n \Leftrightarrow y_n \to z_1$
LEACMD_MAC	0x54	Q.15		u16	Q.15	Q.31	rsvd.	$Z = \sum_{n=0}^{\infty} (x_n * y_n)$	$x_n \Leftrightarrow y_{n+1} \to z_2$
LEACMD_MAC3	0x134	Q.15		u16	Q.15	Q.31	rsvd.	n=0 n=0N-2 1 2	on SCALEDMAC) * SF
LEACMD_SCALEDMAC	0x138	Q.15		u16	Q.15	Q16.15	Q.15)*31

2.2.1 LEACMD__MAC (rMACsf)

Vector X and Vector Y are point wise multiplied and accumulated and written to vector Z of length one using a 16-bit signed expanding fractional multiplication followed by a 32-bit accumulation.

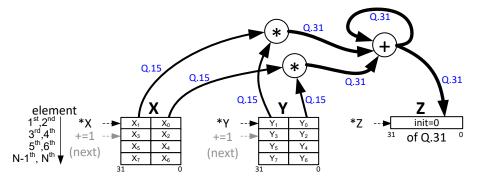
The results saturate to $0x7FFFFFEQ31(1-\varepsilon)$ and 0x8000000EQ31(-1).

Mathematical notation

$$Z = \sum_{n=0}^{N-1} (x_n * y_n)$$

Arguments

pointer to vector X as a 16-bit address value, pointing at an element pair of vector X pointer to vector Y as a 16-bit address value, pointing at an element pair of vector Y pointer to vector Z as a 16-bit address value, pointing to single element vector Z vector size N as a 16-bit unsigned value; defining the number of operations {multiple of 2} **Example** for N = 8



Uses

- FIR filtering of single Q.15 sample (for example, on interrupt driven tasks)
- Single-point correlation, convolution result
- General multiply accumulate purposes

- · Command performs incrementing index order
- The Q15 inputs are expected to be scaled for down to result in a cumulative gain of one to avoid saturation
- · Pad odd vector lengths with zeroes to obtain predictable results



2.2.2 LEACMD__MAC3 (rMAC3sf)

Vector X and Vector Y are point wise multiplied and accumulated. In total three 32 bit results are written to vector Z of length three using a 16-bit signed expanding fractional multiplication followed by a 32-bit accumulation.

The results saturate to $0x7FFFFFEQ31(1-\varepsilon)$ and 0x80000000=Q31(-1).

Mathematical notation

$$z_{0} = \sum_{n=0}^{N-2} (x_{n+1} * y_{n}) \quad \text{with } n = 0 \text{ to (N-2)}$$

$$z_{1} = \sum_{n=0}^{N-1} (x_{n} * y_{n}) \quad \text{with } n = 0 \text{ to (N-1)}$$

$$z_{2} = \sum_{n=0}^{N-2} (x_{n} * y_{n+1}) \quad \text{with } n = 0 \text{ to (N-2)}$$

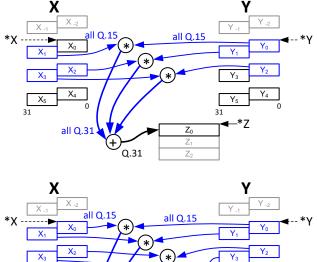
Arguments

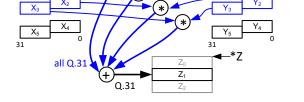
pointer to vector X as a 16-bit address value, pointing at an element pair of vector X

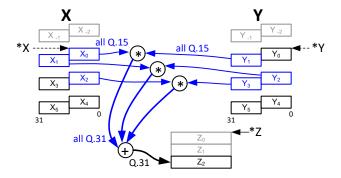
pointer to vector Y as a 16-bit address value, pointing at an element pair of vector Y

pointer to vector Z as a 16-bit address value, pointing to single element vector Z

vector size N as a 16-bit unsigned value defining the length of the tributary terms of Z1 {N must be multiples of 2} **Example** for N=8









2.2.3 LEACMD_SCALEDMAC (sMACsf)

Vector X and Vector Y are point wise multiplied and accumulated and written to vector Z of length one using a 16-bit signed fractional multiplication followed by a 32-bit accumulation and post scaling multiplication.

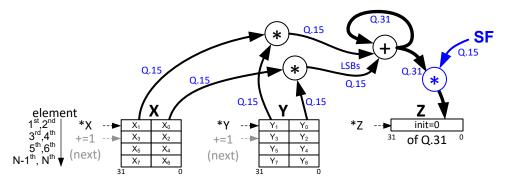
Mathematical notation

 $Z = \left(\sum_{n=0}^{N-1} (x_n \cdot y_n)\right) \cdot SF \qquad \text{with } n = 0 \text{ to (N-1)} \\ \text{and } Z \text{ stored as IQ16.15 (in an 32B/Q.31 container)} \right)$

with n = 0 to (N-1) and Z stored as IQ16.15 (in an 32B/Q.31 container)

Arguments

pointer to vector X as a 16-bit address value, pointing at an element pair of vector X pointer to vector Y as a 16-bit address value, pointing at an element pair of vector Y pointer to vector Z as a 16-bit address value, pointing to single element vector Z scale factor SF as a 16-bit signed Q.15 value to scale down the final result vector size N as a 16-bit unsigned value; defining the number of operations {multiple of 2} **Example** for N=8



Uses

- FIR filtering of single Q.15 sample (for example, on interrupt driven tasks)
- Single-point correlation, convolution result
- General multiply accumulate purposes

Comments

- Command performs incrementing index order
- · The SF gives enough headroom to avoid saturation
- · Pad odd vector lengths with zeroes to obtain predictable results

Comparison With Nonscaled Version (LEACMD_MAC)

On FIRs: In high TAP count FIRs the individual coefficients turn out to be really small. Quantizing the coefficients result into a numeric significance of below 8 bits for some filter characteristics. Here scaling is applied. The SCALEDMAC command has enough mathematical headroom to prevent saturation.

On Correlation and Convolution: To obtain a high resolution on ADCs the signal is amplified to leave only a small amplitude margin. Those signals are often left aligned (LSB's are padded with zeros). This allows algorithms that are independent from the ADC's conversion width. Using the 'SCALEDMAC allows to maintain high numeric significance while avoiding saturation.

Note

The overall filter gain is changed by scaling and must be considered in the operations in Section 2.3.

2.3 Group 3: MAC, Point-Wise FIR, Correlation, and Convolution Operations

Table 2-3 lists the parameters for this group.

Table 2-3. Group 3 Parameters: MAC, Point-Wise FIR, Correlation, and Convolution Operations

		input1	reserved1	vectorSize	input2	output	input1Offset	input2Offset	outputOffset	
		V _{S0} *	occu.	Siz	V _{S1} *	V _D *	V _{s0} +=	V _{S1} +=	V _D +=	
		=& X	ö	=N	=&Y	=& Z	=X _i	=Y _i	=Zi	
LEACMDMACMATRIX	0x100	Q.15		u16	Q.15	Q.31	s16	s16	s16	N-1
LEACMD_MACLONGMATRIX	0xD0	Q.31		u16	Q.31	Q.31	s16	s16	rsvd.	$Z = \sum (x_n * y_n)$
LEACMD_MACCOMPLEXMATRIX	0x68	cQ.15		u16	cQ.15	cQ.31	s16	s16	rsvd.	$\sum_{n=0}^{\infty} (n + y_n)$
LEACMDMACCOMPLEXCONJUGATEMATRIX	0x6C	cQ.15		u16	cQ.15	cQ.31	s16	s16	rsvd.	

2.3.1 LEACMD__MACMATRIX (rMACMsf)

Vector X and Vector Y are point wise multiplied and accumulated and written to vector Z of length one using a 16-bit signed expanding fractional multiplication followed by a 32-bit accumulation.

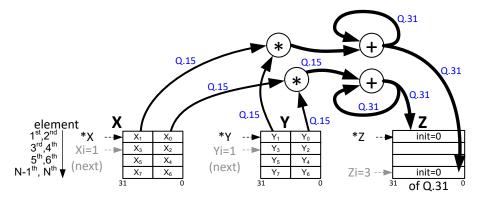
The results saturate to $0x7FFFFFEQ31(1-\epsilon)$ and $0x80000000\equivQ31(-1)$.

Mathematical notation

$$z_0 = \sum_{n=0}^{N-2} (x_n \cdot y_n) \quad \text{with } n = 0, 2, 6 \text{ to } (N-2)$$
$$z_1 = \sum_{n=1}^{N-1} (x_n \cdot y_n) \quad \text{with } n = 1, 3, 5 \text{ to } (N-1)$$

Arguments

pointer to vector Xas a 16-bit address value, pointing at an element pair of vector X pointer to vector Yas a 16-bit address value, pointing at an element pair of vector Y pointer to vector Z as a 16-bit address value, pointing to single element vector Z increment step of Xi as a 16-bit signed value, *X is incremented with (a 1 increments two 16B points) increment step of Yi as a 16-bit signed value, *Y is incremented with (a 1 increments two 16B points) increment step of Zi as a 16-bit signed value, *Z is incremented with for storing odd indexed results vector size N as a 16-bit unsigned value; defining the number of operations {multiple of 2} **Example** for Xi=Yi=1; Zi=3; N=8



Uses

- FIR filtering of single/dual Q.15 sample (for example, on interrupt driven tasks)
- Single or dual point correlation, convolution
- Single or dual scalar dot product in matrix
- General multiply accumulate purposes

- Alternative to an incrementing index order of operations a decrementing order is possible; then starting with
 pointers pointing to last elements
- · Matrices and complex numbers are treated as vectors
- The Q15 inputs are expected to be scaled for down to result in a cumulative gain of one to avoid saturation
- · Pad odd vector lengths with zeroes to obtain predictable results



2.3.2 LEACMD__MACLONGMATRIX (rMACMIf)

Vector X and Vector Y are point wise multiplied and accumulated and written to vector Z of length one using a 31bit signed fractional multiplication followed by a 32-bit accumulation.

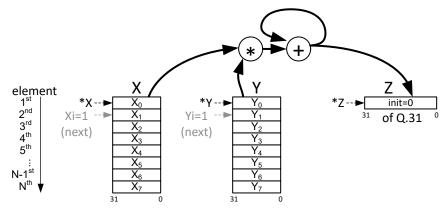
The result saturates to $0x7FFFFFEQ31(1-\varepsilon)$ and 0x8000000EQ31(-1).

Mathematical notation

$$Z = \sum_{n=0}^{N-1} (x_n * y_n) \quad \text{with } n = 0 \text{ to (N-1)}$$

Arguments

pointer to vector X as a 16-bit address value, pointing at an element of vector X pointer to vector Y as a 16-bit address value, pointing at an element of vector Y pointer to vector Z as a 16-bit address value, pointing to single element vector Z increment step of Xi as a 16-bit signed value, *X is incremented with after each operation increment step of Yi as a 16-bit signed value, *Y is incremented with after each operation vector size N as a 16-bit unsigned value; defining the number of operations **Example** for Xi=Yi=1; N=8



Uses

- FIR filtering of single Q.31 sample (for example, on interrupt driven tasks)
- Single point correlation, convolution
- Scalar dot product in matrix
- General multiply accumulate purposes

- Alternative to an incrementing index order of operations a decrementing order is possible; then starting with
 pointers pointing to last elements
- Matrices and complex numbers are treated as vectors
- The Q.31 inputs are expected to be scaled down to result in a cumulative gain of one or avoid saturation

2.3.3 LEACMD__MACCOMPLEXMATRIX (cMACMsf)

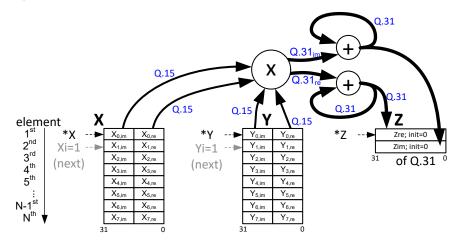
Vector X and Vector Y are point wise multiplied and accumulated and written to vector Z of length one using a 16-bit signed expanding fractional multiplication followed by a 32-bit accumulation. The results saturate to $0x7FFFFFEQ31(1-\epsilon)$ and 0x8000000EQ31(-1).

Mathematical notation

$$z = \sum_{n=0}^{N-1} (x_n \cdot y_n) \text{ with } n = 0 \text{ to (N-1), while}$$
$$z_{re} = \sum_{n=0}^{N-1} (x_{n,re} \cdot y_{n,re} - x_{n,im} \cdot y_{n,im}) \text{ and}$$
$$z_{im} = \sum_{n=0}^{N-1} (x_{n,im} \cdot y_{n,re} + x_{n,re} \cdot y_{n,im})$$

Arguments

pointer to vector X as a 16-bit address value, pointing at an element of vector X pointer to vector Y as a 16-bit address value, pointing at an element of vector Y pointer to vector Z as a 16-bit address value, pointing to single element vector Z increment step of Xi as a 16-bit signed value, *X is incremented with after each operation increment step of Yi as a 16-bit signed value, *Y is incremented with after each operation vector size N as a 16-bit unsigned value; defining the number of operations **Example** for Xi=Yi=1; N=8



Uses

- Complex FIR filtering of single cQ.31 sample (for example, on interrupt driven tasks)
- Phase controlled filtering
- Single side band controlled filtering
- Complex valued data processing
- Trajectory control

- Alternative to an incrementing index order of operations a decrementing order is possible; then starting with pointers pointing to last elements
- The Q.15 inputs are expected to be scaled down to result in a cumulative gain of one or avoid saturation

2.3.4 LEACMD__MACCOMPLEXCONJUGATEMATRIX (cMACMCONJsf)

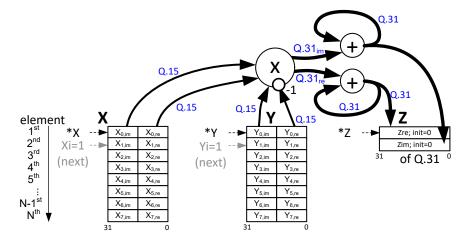
Vector X and Vector Y^{*} are point wise multiplied and accumulated and written to vector Z of length one using a 16-bit signed expanding fractional multiplication followed by a 32-bit accumulation. The results saturate to $0x7FFFFFEQ31(1-\epsilon)$ and 0x8000000EQ31(-1).

Mathematical notation

$$\begin{split} z &= \sum_{n=0}^{N-1} (x_n \cdot y_n^*) \quad \text{with n = 0 to (N-1), while} \\ z_{re} &= \sum_{n=0}^{N-1} (x_{n,re} \cdot y_{n,re} + x_{n,im} \cdot y_{n,im}) \quad \text{and} \\ z_{im} &= \sum_{n=0}^{N-1} (x_{n,im} \cdot y_{n,re} - x_{n,re} \cdot y_{n,im}) \end{split}$$

Arguments

pointer to vector X as a 16-bit address value, pointing at an element of vector X pointer to vector Y as a 16-bit address value, pointing at an element of vector Y pointer to vector Z as a 16-bit address value, pointing to single element vector Z increment step of Xi as a 16-bit signed value, *X is incremented with after each operation increment step of Yi as a 16-bit signed value, *Y is incremented with after each operation vector size N as a 16-bit unsigned value; defining the number of operations **Example** for Xi=Yi=1; N=8



Uses

- Complex FIR filtering of single cQ.31 sample (for example, on interrupt driven tasks)
- Phase controlled filtering
- · Single side band controlled filtering
- · Complex valued data processing
- Trajectory control

- Alternative to an incrementing index order of operations a decrementing order is possible; then starting with
 pointers pointing to last elements
- The Q.15 inputs are expected to be scaled down to result in a cumulative gain of one or avoid saturation

2.4 Group 4: Basic Min/Max Vector Search on 16B Data

Table 2-4 lists the parameter structure for this group.

		input1	reserved1	vectorSize	output	
		V _{s0} * =& X	occu.		D* =& Z	$z_0 = \max_n \{x_0, \dots, x_n\};$ $z_n = n \longrightarrow z_0$
LEACMDMAX	0x104	IQ16		u16	V,P	$z_1 = n \xrightarrow{max} z_0$;
LEACMDMAXUNSIGNED	0x10C	u16		u16	V,P	$z_0 = \min_n \{x_0, \dots, x_n\};$
LEACMDMIN	0x108	IQ16		u16	V,P	
LEACMDMINUNSIGNED	0x110	u16		u16	V,P	$z_1 = n \xrightarrow{min} z_0$;

2.4.1 LEACMD__MAX (rMAXs)

This command returns the value and position of the first maximum of vector X consisting of 16-bit signed numbers using incrementing order from index zero.

Mathematical notation

$$z_0 = \max_{n=0}^{N-1} (x_n \cdot y_n) \quad \text{with } n = 0 \text{ to (N-1), and}$$
$$z_1 = n \xrightarrow[\max]{\max} z_0$$

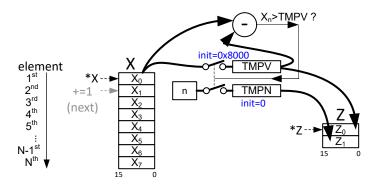
Arguments

pointer to vector X as a 16-bit address value, pointing at an element of vector X

pointer to vector Z as a 16-bit address value, pointing to single element vector Z

vector size N as a 16-bit unsigned value; defining the length of vector {multiple of 2}

Example



Uses

- Search for energy or amplitude peaks in time and spectrum
- · Search for formants
- · General max search

Comments

• With all values the same then the correct value and index=0 is returned (pointing as first element)



2.4.2 LEACMD_MIN (rMINs)

This command returns the value and position of the first minimum of vector X consisting of 16-bit signed numbers using incrementing order from index zero.

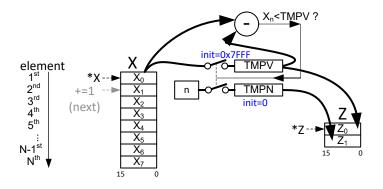
Mathematical notation

$$z_0 = \min_{n=0}^{N-1} (x_n \cdot y_n) \quad \text{with } n = 0 \text{ to (N-1), and}$$
$$z_1 = n \xrightarrow{\min} z_0$$

Arguments

pointer to vector X as a 16-bit address value, pointing at an element of vector X pointer to vector Z as a 16-bit address value, pointing to single element vector Z vector size N as a 16-bit unsigned value; defining the length of vector {multiple of 2}

Example



Uses

- · Search for energy or amplitude notches in time and spectrum
- · General min search

Comments

• With all values the same then the correct value and index=0 is returned (pointing as first element)

2.4.3 LEACMD__MAXUNSIGNED (rMAXus)

This command returns the value and position of the first maximum of vector X consisting of 16-bit unsigned numbers using incrementing order from index zero.

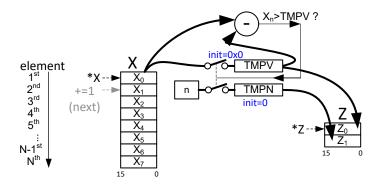
Mathematical notation

$$z_0 = \max_{n=0}^{N-1} (x_n \cdot y_n) \text{ with } n = 0 \text{ to (N-1), and}$$
$$z_1 = n \xrightarrow{\max} z_0$$

Arguments

pointer to vector X as a 16-bit address value, pointing at an element of vector X pointer to vector Z as a 16-bit address value, pointing to single element vector Z vector size N as a 16-bit unsigned value; defining the length of vector {multiple of 2}

Example



Uses

- · search for energy or amplitude peaks in time and spectrum
- search for Formants
- · general max search

Comments

• with all values the same then the correct value and index=0 is returned (pointing as first element)



2.4.4 LEACMD__MINUNSIGNED (rMINus)

This command returns the value and position of the first minimum of vector X consisting of 16-bit unsigned numbers using incrementing order from index zero.

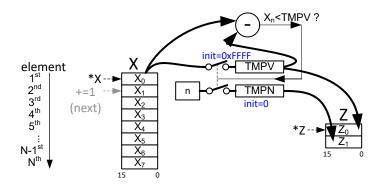
Mathematical notation

$$z_0 = \min_{n=0}^{N-1} (x_n \cdot y_n) \text{ with } n = 0 \text{ to (N-1), and}$$
$$z_1 = n \xrightarrow{\min} z_0$$

Arguments

pointer to vector X as a 16-bit address value, pointing at an element of vector X pointer to vector Z as a 16-bit address value, pointing to single element vector Z vector size N as a 16-bit unsigned value; defining the length of vector {multiple of 2}

Example



Uses

- Search for energy or amplitude notches in time and spectrum
- General min search

Comments

• With all values the same then the correct value and index=0 is returned (pointing as first element)

2.5 Group 5: Generic Min/Max Vector Search on 32B Data

Table 2-5 lists the parameter structure for this group.

		input1	reserved1	vectorSize	output	input1Offset	reserved2	
		V _{S0} *	0	Siz	D*	V _{S0} +=		$z_0 = \max_n \{x_0, \dots, x_n\};$
		=& X	8	=N	=& Z	=X _i		$z_1 = n \xrightarrow{max} z_0$;
LEACMD_MAXLONGMATRIX	0xD8	IQ32		u16	V,P	s16	rsvd.	n=0N-1
LEACMDMAXUNSIGNEDLONGMATRIX	0x114	u32		u16	V,P	s16	rsvd.	$z_0 = \min_n \{x_0, \dots, x_n\};$
LEACMDMINLONGMATRIX	0xDC	IQ32		u16	V,P	s16	rsvd.	$z_1 = n \xrightarrow{min} z_0;$
LEACMDMINUNSIGNEDLONGMATRIX	0x118	u32		u16	V,P	s16	rsvd.	n=0N-1

Table 2-5. Group 5 Parameters: Generic Min/Max Vector Search on 32B Data

2.5.1 LEACMD__MAXLONGMATRIX (rMAXMI)

This command returns the value and position of the first maximum of vector **X** consisting of 32-bit signed numbers.

Mathematical notation

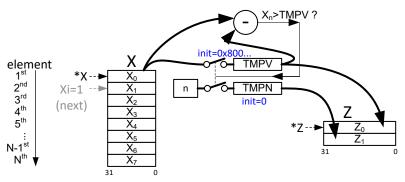
$$z_0 = \max_{n=0}^{N-1} (x_n \cdot y_n) \text{ with } n = 0 \text{ to (N-1), and}$$
$$z_1 = n \xrightarrow[\max]{\max} z_0$$

Arguments

pointer to vector X as a 16-bit address value, pointing at an element of vector X pointer to vector Z as a 16-bit address value, pointing to single element vector Z increment step of Xi as a 16-bit signed value, *X is incremented after each compare

vector size N as a 16-bit unsigned value; defining the length of vector

Example for Xi=1; N=8



Uses

- Search for energy or amplitude peaks in time and spectrum
- Search for formants
- General max search

- Alternative to an incrementing index order of operations a decrementing order is possible; then starting with
 pointers pointing to last elements
- With all values the same then the correct value and index=0 is returned (pointing as first element)



2.5.2 LEACMD__MINLONGMATRIX (rMINMI)

This command returns the value and position of the first minimum of vector **X** consisting of 32-bit signed numbers.

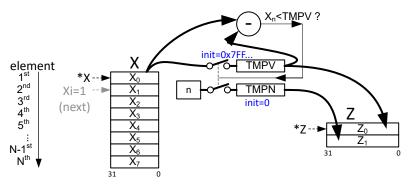
Mathematical notation

$$z_0 = \min_{n=0}^{N-1} (x_n \cdot y_n) \text{ with } n = 0 \text{ to (N-1), and}$$
$$z_1 = n \xrightarrow{\min} z_0$$

Arguments

pointer to vector X as a 16-bit address value, pointing at an element of vector X pointer to vector Z as a 16-bit address value, pointing to single element vector Z increment step of Xi as a 16-bit signed value, *X is incremented after each compare vector size N as a 16-bit unsigned value; defining the length of vector

Example for Xi = 1, N = 8



Uses

- · Search for energy or amplitude notches in time and spectrum
- General min search

- Alternative to an incrementing index order of operations a decrementing order is possible; then starting with pointers pointing to last elements
- With all values the same then the correct value and index=0 is returned (pointing as first element)

2.5.3 LEACMD__MAXUNSIGNEDLONGMATRIX (rMAXMul)

This command returns the value and position of the first maximum of vector X consisting of 16-bit unsigned numbers using incrementing order from index zero.

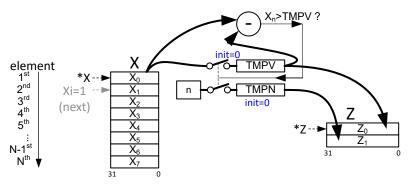
Mathematical notation

$$z_0 = \max_{n=0}^{N-1} (x_n \cdot y_n) \text{ with } n = 0 \text{ to (N-1), and}$$
$$z_1 = n \xrightarrow{\max} z_0$$

Arguments

pointer to vector X as a 16-bit address value, pointing at an element of vector X pointer to vector Z as a 16-bit address value, pointing to single element vector Z increment step of Xi as a 16-bit signed value, *X is incremented after each compare vector size N as a 16-bit unsigned value; defining the length of vector

Example for Xi = 1, N = 8



Uses

- · Search for energy or amplitude peaks in time and spectrum
- Search for formants
- General max search

- Alternative to an incrementing index order of operations a decrementing order is possible; then starting with pointers pointing to last elements
- With all values the same then the correct value and index=0 is returned (pointing as first element)





2.5.4 LEACMD__MINUNSIGNEDLONGMATRIX (rMINMul)

This command returns the value and position of the first minimum of vector X consisting of 32-bit unsigned numbers.

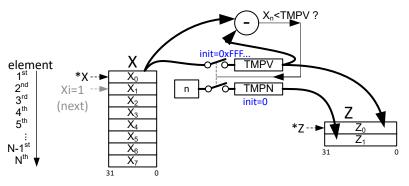
Mathematical notation

$$z_0 = \min_{n=0}^{N-1} (x_n \cdot y_n) \text{ with } n = 0 \text{ to (N-1), and}$$
$$z_1 = n \xrightarrow{\min} z_0$$

Arguments

pointer to vector X as a 16-bit address value, pointing at an element of vector X pointer to vector Z as a 16-bit address value, pointing to single element vector Z increment step of Xi as a 16-bit signed value, *X is incremented after each compare vector size N as a 16-bit unsigned value; defining the length of vector

Example for Xi=1; N=8



Uses

- · Search for energy or amplitude notches in time and spectrum
- General min search

- Alternative to an incrementing index order of operations a decrementing order is possible; then starting with pointers pointing to last elements
- With all values the same then the correct value and index=0 is returned (pointing as first element)

2.6 Group 6: Basic Min/Max Vector Search on Dual 16B Data

Table 2-6 lists the parameter structure for this group.

Table 2-6. Group 6 Parameters: Basic Min/Max Vector Search on Dual 16B Data

		input1	reserved1	FF to		output	reserved2		
		V _{S0} *	ccu.		V _{S0} +=. =Xi	D*		$z_0 = \max_n \{x_0, \dots, x_n\};$	$z_1 = \max_n \{x_1, \dots, x_n\};$
		=&X	0	=N	-4	=& Z		$z_2 = n \xrightarrow{max} z_0$;	$z_3 = n \xrightarrow{max} z_1;$
LEACMDMAXMATRIX	0x44	IQ16		u16	s16	V_E, V_0, P_E, P_0	rsvd.	n=0,2,4N-2	n=1,3,5N-1
LEACMDMAXUNSIGNEDMATRIX	0xE8	u16		u16	s16	V_E, V_O, P_E, P_O	rsvd.	$z_0 = \min_n \{x_0, \dots, x_n\};$	$z_1 = \min_n \{x_1, \dots, x_n\};$
LEACMDMINMATRIX	0x48	IQ16		u16	s16	V_E, V_0, P_E, P_0	rsvd.	$z_2 = n \xrightarrow{min} z_0$;	$z_3 = n \xrightarrow{min} z_1$;
LEACMDMINUNSIGNEDMATRIX	0xEC	u16		u16	s16	V_E, V_O, P_E, P_O	rsvd.	n=0,2,4N-2	n=1,3,5N-1



2.6.1 LEACMD_MAXMATRIX (rMAXMs)

This command returns the values and positions of the first maxima of vector X consisting of 16-bit signed numbers. X consists of two subvectors with even and odd indexed elements.

Mathematical notation

$$z_{0} = \max_{n=0}^{N-2} (x_{n} \cdot y_{n}^{*}) \text{ with } n = 0, 2, 6 \text{ to (N-2), and}$$

$$z_{2} = n \xrightarrow[n=1]{\max} z_{0}$$

$$z_{1} = \max_{n=1}^{N-1} (x_{n} \cdot y_{n}^{*}) \text{ with } n = 1, 3, 5 \text{ to (N-1), and}$$

$$z_{3} = n \xrightarrow[m=1]{\max} z_{1}$$

Arguments

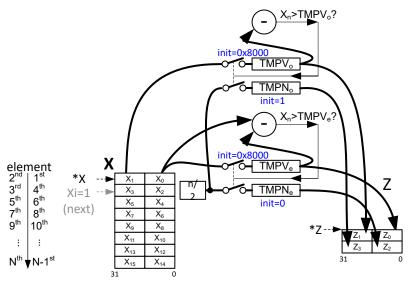
pointer to vector X as a 16-bit address value, pointing at an element pair of vector X

pointer to vector Z as a 16-bit address value, pointing to vector Z

increment step of Xi as a 16-bit signed value, *X is incremented (pairs)

vector size N as a 16-bit unsigned value; defining the length of vector{multiple of 2}

Example for Xi=1; N=16



Uses

- Search for energy or amplitude peaks in time and spectrum
- · General max search on dual vectors
- General max search on SIMD vectors

- Alternative to an incrementing index order of operations a decrementing order is possible; then starting with
 pointers pointing to last elements
- With all values the same then the correct value and index=0 is returned (pointing as first element pair)



2.6.2 LEACMD__MINMATRIX (rMINMs)

This command returns the values and positions of the first minima of vector X consisting of 16-bit signed numbers. X consists of two sub vector with even and odd indexed elements.

Mathematical notation

$$z_{0} = \min_{n=0}^{N-2} (x_{n} \cdot y_{n}^{*}) \text{ with } n = 0, 2, 6 \text{ to (N-2), and}$$

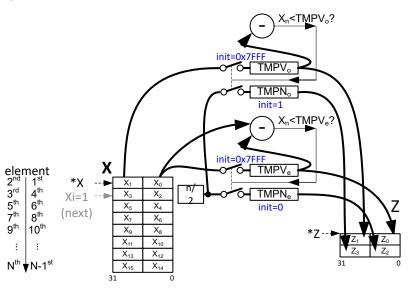
$$z_{2} = n \xrightarrow{\text{min}} z_{0}$$

$$z_{1} = \min_{n=1}^{N-1} (x_{n} \cdot y_{n}^{*}) \text{ with } n = 1, 3, 5 \text{ to (N-1), and}$$

$$z_{3} = n \xrightarrow{\text{min}} z_{1}$$

Arguments

pointer to vector X as a 16-bit address value, pointing at an element of vector X pointer to vector Z as a 16-bit address value, pointing to single element vector Z increment step of Xi as a 16-bit signed value, *X is incremented (pairs) vector size N as a 16-bit unsigned value; defining the length of vector {multiple of 2} **Example** for Xi=1; N=16



Uses

- Search for energy or amplitude notches in time and spectrum
- · General min search on dual vectors
- General min search on SIMD vectors

- Alternative to an incrementing index order of operations a decrementing order is possible; then starting with
 pointers pointing to last elements
- With all values the same then the correct value and index=0 is returned (pointing as first element pair)



2.6.3 LEACMD_MAXUNSIGNEDMATRIX (rMAXMus)

This command returns the value and position of the first maximum of vector X consisting of 16-bit unsigned numbers using incrementing order from index zero.

Mathematical notation

$$z_{0} = \max_{n=0}^{N-2} (x_{n} \cdot y_{n}^{*}) \text{ with } n = 0, 2, 6 \text{ to (N-2), and}$$

$$z_{2} = n \xrightarrow[n=1]{\max} z_{0}$$

$$z_{1} = \max_{n=1}^{N-1} (x_{n} \cdot y_{n}^{*}) \text{ with } n = 1, 3, 5 \text{ to (N-1), and}$$

$$z_{3} = n \xrightarrow[m=1]{\max} z_{1}$$

Arguments

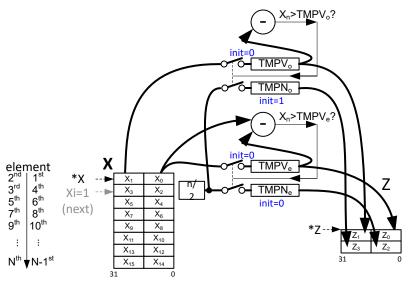
pointer to vector X as a 16-bit address value, pointing at an element of vector X

pointer to vector Z as a 16-bit address value, pointing to vector Z

increment step of Xi as a 16-bit signed value, *X is incremented (pairs)

vector size N as a 16-bit unsigned value; defining the length of vector {multiple of 2}

Example for Xi=1; N=16



Uses

- Search for energy or amplitude peaks in time and spectrum
- · General max search on dual vectors
- General max search on SIMD vectors

- Alternative to an incrementing index order of operations a decrementing order is possible; then starting with
 pointers pointing to last elements
- With all values the same then the correct value and index=0 is returned (pointing as first element pair)



2.6.4 LEACMD___MINUNSIGNEDMATRIX (rMINMus)

This command returns the value and position of the first minimum of vector X consisting of 16-bit unsigned numbers. X consists of two sub vector with even and odd indexed elements.

Mathematical notation

$$z_{0} = \min_{n=0}^{N-2} (x_{n} \cdot y_{n}^{*}) \text{ with } n = 0, 2, 6 \text{ to (N-2), and}$$

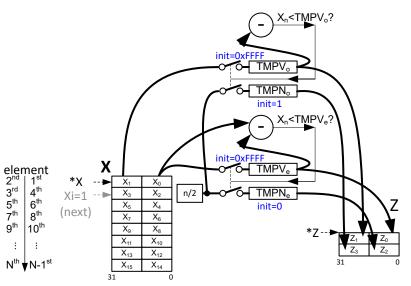
$$z_{2} = n \xrightarrow{\text{min}} z_{0}$$

$$z_{1} = \min_{n=1}^{N-1} (x_{n} \cdot y_{n}^{*}) \text{ with } n = 1, 3, 5 \text{ to (N-1), and}$$

$$z_{3} = n \xrightarrow{\text{min}} z_{1}$$

Arguments

pointer to vector X as a 16-bit address value, pointing at an element of vector X pointer to vector Z as a 16-bit address value, pointing to single element vector Z increment step of Xi as a 16-bit signed value, *X is incremented (pairs) vector size N as a 16-bit unsigned value; defining the length of vector {multiple of 2} **Example** for Xi = 1, N = 16



Uses

- Search for energy or amplitude notches in time and spectrum
- · General min search on dual vectors
- General min search on SIMD vectors

- Alternative to an incrementing index order of operations a decrementing order is possible; then starting with
 pointers pointing to last elements
- With all values the same then the correct value and index=0 is returned (pointing as first element pair)

2.7 Group 7: Block FIR, Correlation, Convolution Functions

Table 2-7 lists the parameter structure for this group.

		= % * input1	occu. reserved1	<u>Z Ri</u> vectorSize	vcefs	= s= output	H an filterLength	se circularBufferMask	였 scaleFactor	
LEACMD FIR		Q.15	0	u16	Q.15	Q.15	u16	u16	rsvd.	K-1
LEACMD_FIRLONG	0xC8			u16	Q.31	Q.31	u16	u16	rsvd.	$z_n = \sum (h_k * x_{K-k+n-1});$
LEACMDFIRCOMPLEX	0xE0	cQ.15		u16	cQ.15	cQ.15	u16	u16	rsvd.	$\sum_{k=0}^{n} \sum_{k=0}^{n} \sum_{k$
LEACMDFIRCOMPLEXLONG	0x124	cQ.31		u16	cQ.31	cQ.31	u16	u16	rsvd.	n=0N-1 (for mask=0xff)
LEACMD_SCALEDFIR	0x13C	Q.15		u16	Q.15	Q.15	u16	u16	Q.15) * SF on SCALEDFIR

Table 2-7. Group 7 Parameters: Block FIR, Correlation, Convolution Functions



2.7.1 LEACMD__FIR (rFIRsf)

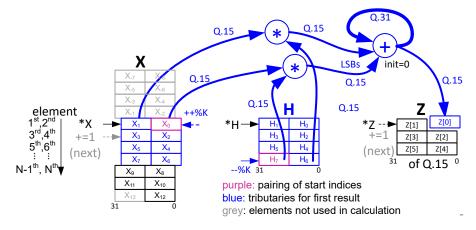
Vector X and vector H are convolved to vector Z using using16 bit fractional multiplications followed by a 32-bit accumulation. The results are stored as 16-bit numbers.

Mathematical notation

$$z_n = \sum_{k=0}^{K-1} (h_k \cdot x_{K-k+n-1})$$
 with n = 0 to (N-1) and k = 0 to (K-1)

Arguments

pointer to vector X as a 16-bit address value, pointing at an element pair of vector X pointer to vector H as a 16-bit address value, pointing at an element pair of vector H pointer to vector Z as a 16-bit address value, pointing at an element pair of vector Z Length K as a 16-bit unsigned value; defining the length/width of the convolution {multiple of 2} vector size N as a 16-bit unsigned value; defining the length of vector {multiple of 2} address mask as a 16-bit pattern used for circular address generation for vector X **Example** for K = 8, N = 6, mask = 0xFFFF



Uses

- Real valued convolution of 16-bit vectors
- Real valued correlation of 16-bit vectors (if vector H is pointwise reversed)
- Real valued FIR filtering of Q.15 vectors or streams

Comments

- From vector X elements with index K+N are used for the result calculation
- The input vector X supports circular addressing to support samples streams
- The output values are truncated to 16-bits (use proper coefficients to avoid numeric misrepresentation)

Special application case for FIR (finite impulse response) filter

A convolution is one method to implement an FIR filter; Here the arguments are used as follows.

Arguments

pointer to vector X as a 16-bit address value, pointing to element pair of oldest sample.

pointer to vector H as a 16-bit address value, pointing to start of transfer function H

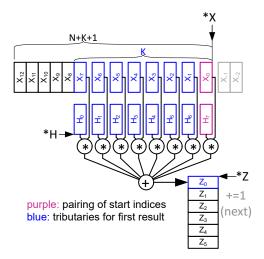
pointer to vector Z as a 16-bit address value, pointing to start of next output sample pair Z

Length K as a 16-bit unsigned value; count of filter TAPs {multiple of 2}

vector size N as a 16-bit unsigned value; defining the number of output pairs to calculate

address mask as a 16-bit pattern used for circular address generation for vector X

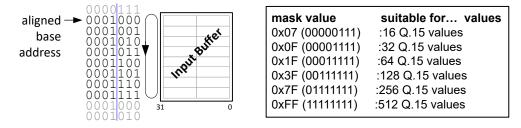
Example for K = 8, N = 6, mask = 0xFFFF



Circular addressing feature of the input buffer

Specific values of argument *mask* cause the address generator for input vector *X to generate circular addresses. If the lower three bits are set to "one" then only the lower three bits count normally the upper address bits remain static. At the end of a block of eight an address wrap to the base address occurs. The base address of such a circular buffer must be aligned to an address boundary with a modulo-value of zero: *mod(address, blocksize)= zero*. When the circular buffers are used then the input data provision is also done in a circular addressed manner.

address mask:0000111 (for 32 bit words)





2.7.2 LEACMD__FIRLONG (rFIRIf)

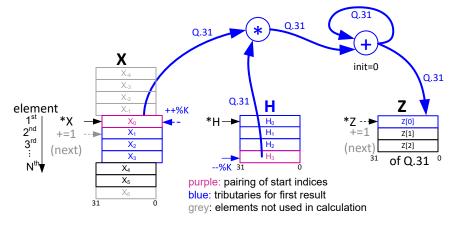
Vector X and vector H are convolved to vector Z using using32 bit fractional multiplications followed by a 32-bit accumulation. The results are stored as 32-bit numbers.

Mathematical notation

$$z_n = \sum_{k=0}^{K-1} (h_k \cdot x_{K-k+n-1})$$
 with n = 0 to (N-1) and k = 0 to (K-1)

Arguments

pointer to vector X as a 16-bit address value, pointing at an element of vector X pointer to vector H as a 16-bit address value, pointing at an element of vector H pointer to vector Z as a 16-bit address value, pointing at an element of vector Z Length K as a 16-bit unsigned value, defining the length/width of the convolution vector size N as a 16-bit unsigned value, defining the length of vector address mask as a 16-bit pattern used for circular address generation for vector X **Example** for K = 4, N = 3, mask = 0xFFFF



Uses

- Real valued convolution of 32-bit vectors
- Real valued correlation of 32-bit vectors (if vector H is point wide reversed)i
- Real valued FIR filtering of Q.31 vectors/streams

Comments

- · From vector X elements with index K+N-1 are used for the result calculation
- The input vector X supports circular addressing to support samples streams
- The output values are truncated to 31 bits (use proper coefficients to avoid numeric misrepresentation)

Special application case for FIR (finite impulse response) filter

A convolution is one method to implement an FIR filter, Here the arguments are used as follows.

Arguments

pointer to vector X as a 16-bit address value, pointing to element of oldest sample.

pointer to vector H as a 16-bit address value, pointing to start of transfer function H

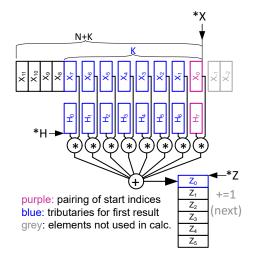
pointer to vector Z as a 16-bit address value, pointing to start of next output sample Z

Length K as a 16-bit unsigned value, count of filter TAPs

vector size N as a 16-bit unsigned value, defining the number of output pairs to calculate

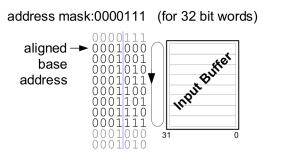
address mask as a 16-bit patter used for circular address generation for vector X

Example for K = 8, N = 6, mask = 0xFFFF



Circular addressing feature of the input buffer

Specific values of argument *mask* cause the address generator for input vector *X to generate circular addresses. If the lower three bits are set to "one" then only the lower three bits count normally the upper address bits remain static. At the end of a block of eight an address wrap to the base address occurs. The base address of such a circular buffer must be aligned to an address boundary with a modulo-value of zero. *mod(address, blocksize) = zero*. When the circular buffers are used then the input data provision is also done in a circular addressed manner.



mask value	suitable for values
0x07 (00000111)	eight Q.31 values:
0x0F (00001111)	:16 Q.31 values
0x1F (00011111)	:32 Q.31 values
0x3F (00111111)	:64 Q.31 values
0x7F (01111111)	:128 Q.31 values
0xFF (11111111)	:256 Q.31 values
	0x07 (00000111) 0x0F (00001111) 0x1F (00011111) 0x3F (00111111) 0x7F (0111111)



2.7.3 LEACMD__FIRCOMPLEX (cFIRsf)

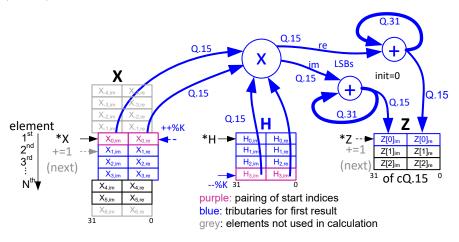
Complex valued vectors X and H are convolved to complex valued vector Z using using16 bit complex fractional multiplications followed by a 32-bit accumulation. The results are stored as 16-bit complex numbers.

Mathematical notation

$$\begin{split} z_n &= \sum_{k=0}^{K-1} (h_k \cdot x_{K-k+n-1}) & \text{ with n = 0 to (N-1) and k=0 to (K-1), while} \\ z_{n,re} &= \sum_{k=0}^{K-1} (h_{k,re} \cdot x_{K-k+n-1,re} - h_{k,im} \cdot x_{K-k+n-1,im}) & \text{ and} \\ z_{n,im} &= \sum_{k=0}^{K-1} (h_{k,im} \cdot x_{K-k+n-1,re} + h_{k,re} \cdot x_{K-k+n-1,im}) \end{split}$$

Arguments

pointer to vector X as a 16-bit address value, pointing at an element of vector X pointer to vector H as a 16-bit address value, pointing at an element of vector H pointer to vector Z as a 16-bit address value, pointing at an element of vector Z Length K as a 16-bit unsigned value, defining the length/width of the convolution vector size N as a 16-bit unsigned value, defining the length of vector address mask as a 16-bit pattern used for circular address generation for vector X **Example** for K = 4, N = 3, mask = 0xFFFF



Uses

- Complex valued convolution of 16-bit vectors
- · Complex valued correlation of 16-bit vectors (if vector H is point wide reversed)i
- Complex valued FIR filtering of cQ.15 vectors/streams (phased controlled)
- Single side-band controlled filtering of cQ.15 vectors

Comments

- · From vector X elements with index K+N-1 are used for the result calculation
- The input vector X supports circular addressing to support samples streams
- The output values are truncated to 16-bits (use proper coefficients to avoid numeric misrepresentation)



Special application case for FIR (finite impulse response) filter

A convolution is one method to implement an FIR filter, Here the arguments are used as follows.

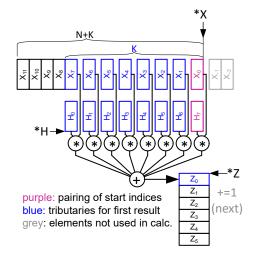
Arguments

pointer to vector X as a 16-bit address value, pointing to element of oldest sample. pointer to vector H as a 16-bit address value, pointing to start of transfer function H pointer to vector Z as a 16-bit address value, pointing to start of next output sample Z Length K as a 16-bit unsigned value, count of filter TAPs

vector size N as a 16-bit unsigned value, defining the number of output pairs to calculate

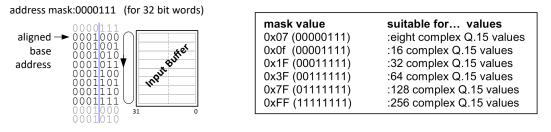
address mask as a 16-bit patter used for circular address generation for vector X

Example for K = 8, N = 6, mask = 0xFFFF



Circular addressing feature of the input buffer

Specific values of argument *mask* cause the address generator for input vector *X to generate circular addresses. If the lower three bits are set to "one" then only the lower three bits count normally the upper address bits remain static. At the end of a block of eight an address wrap to the base address occurs. The base address of such a circular buffer must be aligned to an address boundary with a modulo-value of zero: *mod(address, blocksize) = zero*. When the circular buffers are used then the input data provision is also done in a circular addressed manner.





2.7.4 LEACMD__FIRCOMPLEXLONG (cFIRIf)

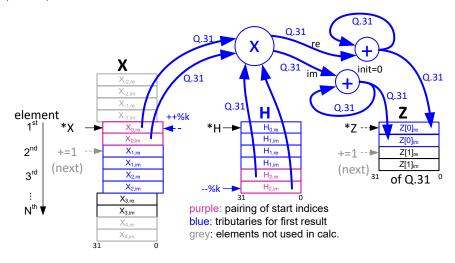
Complex values vectors X and H are convolved to complex valued vector Z using 32-bit complex fractional multiplications followed by a 32-bit accumulation. The results are stored as 32-bit complex numbers.

Mathematical notation

$$\begin{split} z_n &= \sum_{k=0}^{K-1} (h_k \cdot x_{K-k+n-1}) & \text{ with n = 0 to (N-1) and k=0 to (K-1), while} \\ z_{n,re} &= \sum_{k=0}^{K-1} (h_{k,re} \cdot x_{K-k+n-1,re} - h_{k,im} \cdot x_{K-k+n-1,im}) & \text{ and } \\ z_{n,im} &= \sum_{k=0}^{K-1} (h_{k,im} \cdot x_{K-k+n-1,re} + h_{k,re} \cdot x_{K-k+n-1,im}) \end{split}$$

Arguments

pointer to vector X as a 16-bit address value, pointing at an element of vector X pointer to vector H as a 16-bit address value, pointing at an element of vector H pointer to vector Z as a 16-bit address value, pointing at an element of vector Z length K as a 16-bit unsigned value, defining the length/width of the convolution vector size N as a 16-bit unsigned value, defining the length of vector address mask as a 16-bit patter used for circular address generation for vector X **Example** for K = 3, N = 2, mask = 0xFFFF



Uses

- Complex valued convolution of 32-bit vectors
- · Complex valued correlation of 32-bit vectors (if vector H is point wide reversed)i
- Complex valued FIR filtering of cQ.31 vectors/streams (phased controlled)
- Single side-band controlled filtering of cQ.31 vectors

Comments

- From vector X elements with index K+N-1 are used for the result calculation
- The input vector X supports circular addressing to support samples streams
- The output values are truncated to 16-bits (use proper coefficients to avoid numeric misrepresentation)



Special application case for FIR (finite impulse response) filter

A convolution is one method to implement an FIR filter, Here the arguments are used as follows.

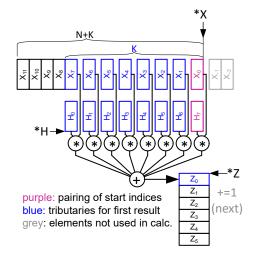
Arguments

pointer to vector X as a 16-bit address value, pointing to element of oldest sample. pointer to vector H as a 16-bit address value, pointing to start of transfer function H pointer to vector Z as a 16-bit address value, pointing to start of next output sample Z Length K as a 16-bit unsigned value, count of filter TAPs

vector size N as a 16-bit unsigned value, defining the number of output pairs to calculate

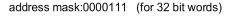
address mask as a 16-bit patter used for circular address generation for vector X

Example for K = 8, N = 6, mask = 0xFFFF



Circular addressing feature of the input buffer

Specific values of argument *mask* cause the address generator for input vector *X to generate circular addresses. If the lower three bits are set to "one" then only the lower three bits count normally the upper address bits remain static. At the end of a block of eight an address wrap to the base address occurs. The base address of such a circular buffer must be aligned to an address boundary with a modulo-value of zero: *mod(address, blocksize) = zero*. When the circular buffers are used then the input data provision is also done in a circular addressed manner.



aligned → 00001111 base 0001001 address 0001010 0001100 0001100 0001101 0001110 0001110 0001100 0001111 0001000 31 0	mask value 0x07 (00000111) 0x0f (00001111) 0x1F (00011111) 0x3F (00111111) 0x7F (01111111) 0xFF (11111111)	suitable for values :four complex Q.31 values :eight complex Q.31 values :16 complex Q.31 values :32 complex Q.31 values :64 complex Q.31 values :128 complex Q.31 values
---	--	---

2.7.5 LEACMD_SCALEDFIR (sFIRsf)

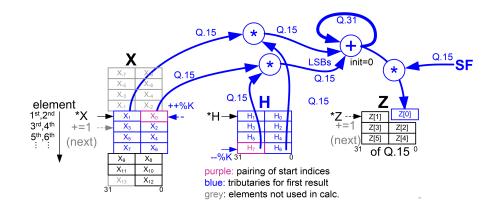
Vector X and vector H are convolved to vector Z using using16 bit fractional multiplications followed by a 32-bit accumulation. The results are stored as 16-bit numbers.

Mathematical notation

$$z_n = \left(\sum_{k=0}^{K-1} (h_k \cdot x_{K-k+n-1})\right) \cdot SF$$
 with n = 0 to (N-1) and k = 0 to (K-1) and
Z_n stored as IQ16.15 (in 32B/Q.31 container)

Arguments

pointer to vector X as a 16-bit address value, pointing at an element pair of vector X pointer to vector H as a 16-bit address value, pointing at an element pair of vector H pointer to vector Z as a 16-bit address value, pointing at an element pair of vector Z Length K as a 16-bit unsigned value, defining the length/width of the convolution {multiple of 2} scale factor SF as a 16-bit signed Q.15 value to scale down the final results vector size N as a 16-bit unsigned value, defining the length of vector {multiple of 2} address mask as a 16-bit pattern used for circular address generation for vector X **Example** for K = 8, N = 6, mask = 0xFFFF



Uses

- · Real valued convolution of 16-bit vectors
- Real valued correlation of 16-bit vectors (if vector H is point wide reversed)i
- Real valued FIR filtering of Q.15 vectors/streams

Comments

- From vector X elements with index K+N are used for the result calculation
- The input vector X supports circular addressing to support samples streams
- The output values are truncated to 16-bits (use proper coefficients to avoid numeric misrepresentation)

Comparison with nonscaled version (LEACMD_FIR)

On FIR: In high TAP count FIRs the individual coefficients turn out to be really small. Quantizing the coefficients result into a numeric significance of below 8 bits for some filter characteristics. Here scaling is applied. The SCALEDFIR command has enough mathematical headroom to prevent saturation.

On Correlation and Convolution: To obtain a high resolution on ADCs, the signal is amplified to leave only a small amplitude margin. Those signals are often left aligned (LSBs are padded with zeros). This allows algorithms that are independent from the ADC conversion width. Using the SCALEDFIR allows to maintain high numeric significance while avoiding saturation.

2.8 Group 8: Taylor McLaurin Functions on Pointwise Vectors or Matrices

Table 2-8 lists the parameter structure for this group.

	input1	reserved1	vectorSize	output	coefs	order		scaleFactor	
	V _{S0} *	cu.	Siz	V _D *	V _c *	Ord			<u>M-1</u>
	=& X	Ö	=N	=& Z	+& C	=M		SF	$z_n = SF \sum (C_m * x_n^m)$
LEASCCMD_POLYNOMIAL 0x2	Q.15		u16	Q.15	Q.15	u16	IQ [`]	16.15	
LEASCCMD_POLYNOMIALLONG 0x3	Q.31		u16	Q.31	Q.31	u16	s16	rsvd.	m=0 n=0N-1



2.8.1 LEACMD_POLYNOMIAL (rPOLYsf)

Polynomials results of vector X and Vector C are written to vector Z using a 16-bit signed fractional multiplication of powers of X and pointwise values of C followed by a 32-bit accumulation and a scaling multiplication.

Mathematical notation

$$z_{n} = \left(\sum_{m=0}^{M-1} (C_{m} \cdot x_{n}^{m})\right) \cdot SF \quad \text{with } n = 0 \text{ to (N-1) and } m = 0 \text{ to (M-1), likewise}$$
$$z_{n} = \left(c_{0} + c_{1} \cdot x_{n} + c_{2} \cdot x_{n}^{2} + c_{3} \cdot x_{n}^{3} + c_{4} \cdot x_{n}^{4} \dots\right) \cdot SF \quad \text{with } n = 0 \text{ to (N-1)}$$

Arguments

pointer to vector X as a 16-bit address value, pointing at an element of vector X

pointer to vector C as a 16-bit address value, pointing to the element pair of vector C with the highest indices (this vector is reverse ordered)

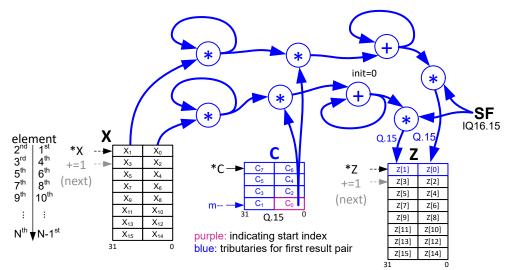
pointer to vector Z as a 16-bit address value, pointing at an element of vector Z

Order M as a 16-bit unsigned value, defining the maximum order of the powers (multiple of two)

vector size N as a 16-bit unsigned value, defining the length of vector (multiple of two)

scaling factor SF as a 32-bit signed value used for post scaling in IQ16.15 format

Example for M = 8, N = 16



Uses

- · Mathematical functions based on Taylor/McLaurin series
- Approximation of nonlinear function using polynomial coefficient fitting methods
- Sensor linearization

Comments

- Some real applications problems require transposition and scaling of the polynomial coefficients to stay within its convergence range
- Coefficient sets for some common functions are provide within the LEA coefficient ROM

Note: Pad the highest-order coefficient with zeros if the polynomial must be of odd order.



2.8.2 LEACMD_POLYNOMIALLONG (rPOLYIf)

Polynomial results of vector X and vector C are written to vector Z using a 32-bit signed fractional multiplication of powers of X and pointwise values of C followed by a 32-bit accumulation and a scaling multiplication.

Mathematical notation

$$z_{n} = \left(\sum_{m=0}^{M-1} (C_{m} \cdot x_{n}^{m})\right) \cdot SF \quad \text{with } n = 0 \text{ to (N-1) and } m=0 \text{ to (M-1), likewise}$$
$$z_{n} = \left(c_{0} + c_{1} \cdot x_{n} + c_{2} \cdot x_{n}^{2} + c_{3} \cdot x_{n}^{3} + c_{4} \cdot x_{n}^{4} \dots\right) \cdot SF \quad \text{with } n = 0 \text{ to (N-1)}$$

Arguments

pointer to vector X as a 16-bit address value, pointing at an element of vector X

pointer to vector C as a 16-bit address value, pointing to the element of vector C with highest index (this vector is reverse ordered)

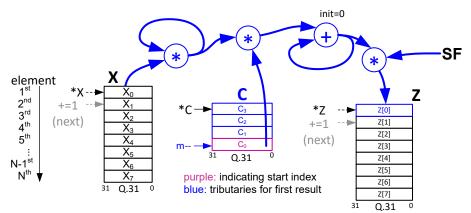
pointer to vector Z as a 16-bit address value, pointing at an element of vector Z

Order M as a 16-bit unsigned value, defining the maximum order of the powers

vector size N as a 16-bit unsigned value, defining the length of vector

scaling factor SF as a 32-bit signed value used for post scaling

Example for M = 4, N = 8



Uses

- · Mathematical functions based on Taylor/McLaurin series
- · Approximation of nonlinear function using polynomial coefficient fitting methods
- Sensor linearization

Comments

- Some real applications problems require transposition and scaling of the polynomial coefficients to stay within its convergence range
- Coefficient sets for some common functions are provide within the LEA coefficient ROM

2.9 Group 9: Fast Fourier Transforms and Inverses

Table 2-9 lists the parameter structure for this group.

		* input1	u. reserved1	<u>o</u> NectorSizeBy2	log2Size		
		=& X	occu.		lb(N)		
LEACMDFFTCOMPLEXAUTOSCALING	0xC0	cQ.15		u16	u16		
LEACMDFFTCOMPLEXFIXEDSCALING	0x10	cQ.15		u16	u16	$X = FFT(x_0,, x_{N-1})$ with	SF=> REG _{DST}
LEACMDFFTCOMPLEXLONG	0x9C	cQ.31		u16	u16	bit reverse sorted input	on AUTOSCALING

Table 2-9. Group 9 Parameters: Fast Fourier Transforms and Inverses



2.9.1 LEACMD__FFTCOMPLEXAUTOSCALING (cFFTsf_as)

Complex vector X is transformed from time domain into frequency domain by performing a decimate-in-time (DIT) Cooley-Tukey Fast-Fourier-Transform operation (FFT). This command is used for real valued and complex valued FFTs. The input values are expected to be Q.15 number pairs in bit-reversed order. The output values are in straight order with signed index. This command performs an "In-Place" operation (output values are stored back to the location of the input values). This command performs automatic scaling to maintain the highest accuracy possible with the given numeric input. The number of required scaling steps is returned in the LEASCPMDST register. The vector X must be address aligned to an address boundary with a modulo-value of zero: *mod(address, blocksize) = zero*.

Use LEASCCMD_BITREVERSECOMPLEX commands for input value shuffling if values are in time linear order.

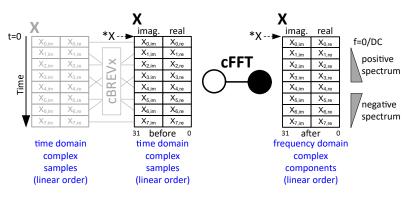
Arguments for complex-valued FFT

pointer to vector X as a 16-bit address value, pointing to first location of aligned vector X

half size N/2 a 16 bit unsigned value, defining half number of complex points (for cFFT16 set to 8, for cFFT128 set to 64, and so on)

log size lb(N) a 16 bit unsigned number, defining the number of Butterfly-Operation slices (for cFFT 16 set to 4, for cFFT128 set to 7, and so on)

Example for cFFT8



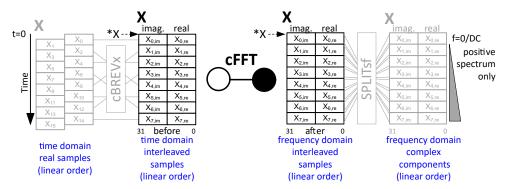
Arguments for real-valued FFT (requires post operation to split real values)

pointer to vector X as a 16-bit address value, pointing to first location of aligned vector X

half size N/2 a 16 bit unsigned value, defining half number of complex points likewise a quarter od real points (for rFFT32 set to 8, for rFFT256 set to 64, and so on)

log size lb(N) a 16 bit unsigned number, defining the number of Butterfly-Operation slices (for rFFT32 set to 4, for rFFT256 set to 7, and so on)

Example for rFFT16



Comments



• An inverse-FFT (iFFT) operation is achieved by feeding the complex spectrum into a FFT. The result is a signal in the time domain running backward in time.



2.9.2 LEACMD__FFTCOMPLEXFIXSCALING (cFFTsf_fs)

Complex vector X is transformed from time domain into frequency domain by performing a decimate-in-time (DIT) Cooley-Tukey Fast-Fourier-Transform (FFT) operation. This command is used for real valued and complex valued FFTs. The input values are expected to be Q.15 number pairs in bit reversed order. The output values are in straight order with signed index. This command performs an "In-Place" operation (output values are stored back to the location of the input values). This Command performs fix scaling to provide result that are independent of the numeric input. The vector X must be address aligned to an address boundary with a modulo-value of zero: *mod(address, blocksize) = zero*.

Use LEASCCMD_BITREVERSECOMPLEX commands for input value shuffling if values are in time linear order.

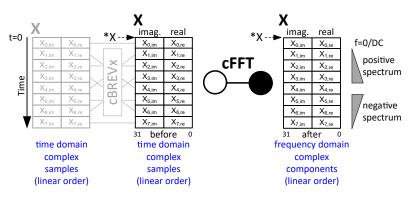
Arguments for complex-valued FFT

pointer to vector X as a 16-bit address value, pointing to first location of aligned vector X

half size N/2 a 16-bit unsigned value, defining half number of complex points (for cFFT16 set to 8, for cFFT128 set to 64, and so on)

log size lb(N) a 16-bit unsigned number, defining the number of Butterfly-Operation slices (for cFFT 16 set to 4, for cFFT128 set to 7, and so on)

Example for cFFT8



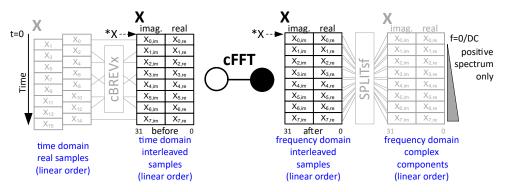
Arguments for real-valued FFT (requires post operation to split real values)

pointer to vector X as a 16-bit address value, pointing to first location of aligned vector X

half size N/2 a 16-bit unsigned value, defining half number of complex points likewise a quarter od real points (for rFFT32 set to 8, for rFFT256 set to 64, and so on)

log size lb(N) a 16-bit unsigned number, defining the number of Butterfly-Operation slices (for rFFT32 set to 4, for rFFT256 set to 7, and so on)

Example for rFFT16



Comments

• An inverse-FFT (iFFT) operation is achieved by feeding the complex spectrum into a FFT. The result is a signal in time-domain running backwards in time.



2.9.3 LEACMD__FFTCOMPLEXLONG (cFFTIf)

Complex vector X is transformed from time domain into frequency domain by performing a decimate-in-time (DIT) Cooley-Tukey Fast-Fourier-Transform operation (FFT). This command is used for real valued and complex valued FFTs. The input values are expected to be Q.31 number pairs in bit reversed order. The output values are in straight order with signed index. This command in performs an "In-Place" operation (output values are stored back to the location of the input values). This Command performs no scaling. A 16pt cFFT therefore has a numeric gain of 16 (scale down the input accordingly). The vector X must be address aligned to an address boundary with a modulo-value of zero: *mod(address, blocksize) = zero*.

Use LEASCCMD_BITREVERSECOMPLEX commands for input value shuffling if values are in time linear order.

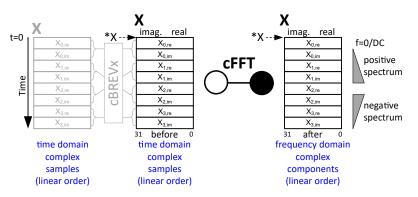
Arguments for complex-valued FFT

pointer to vector X as a 16-bit address value, pointing to first location of aligned vector X

half size N/2 a 16-bit unsigned value, defining half number of complex points (for cFFT16 set to 8, for cFFT128 set to 64, and so on)

log size lb(N) a 16-bit unsigned number, defining the number of Butterfly-Operation slices (for cFFT 16 set to 4, for cFFT128 set to 7, and so on)

Example for cFFT4



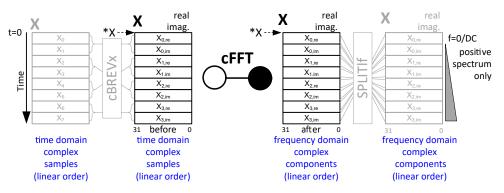
Arguments for real-valued FFT (requires post operation to split real values)

pointer to vector X as a 16-bit address value, pointing to first location of aligned vector X

half size N/2 a 16-bit unsigned value, defining half number of complex points likewise a quarter od real points (for rFFT32 set to 8, for rFFT256 set to 64, and so on)

log size lb(N) a 16-bit unsigned number, defining the number of Butterfly-Operation slices (for rFFT32 set to 4, for rFFT256 set to 7, and so on)

Example for rFFT8



Comments

• An inverse-FFT (iFFT) operation is achieved by feeding the complex spectrum into a FFT. The result is a signal in time-domain running backwards in time.

2.10 Group 10: Bit Reversed Pre-Sort Functions for FFTs

Table 2-10 lists the parameter structure for this group.

Table 2-10. Group 10 Parameters: Bit Reversed Pre-Sort Functions for FFTs

	input1	reserved1	sqryVectorSize	reserved2	
	V _{S0} * =& X	occu.	\sqrt{N}		
LEACMDBITREVERSECOMPLEXEVEN	0x84 cQ.15		u16	rsvd.	$X = bit - reverse(x_0, \dots, x_{N-1})$
LEACMDBITREVERSECOMPLEXODD	0x88 cQ.15		u16	rsvd.	n=0N-1 with
LEACMDBITREVERSECOMPLEXLONGEVEN	0xAC cQ.31		u16	rsvd.	$n+=c \Longrightarrow c \Longrightarrow c \Longrightarrow c \Longrightarrow c \Longrightarrow$
LEACMDBITREVERSECOMPLEXLONGODD	0xB4 cQ.31		u16	rsvd.	$n+= \ c \underset{Bit_{15}}{\Longrightarrow} \dots \ c \underset{Bit_{\sqrt{N}+1}}{\Longrightarrow} c \underset{Bit_{\sqrt{N}}}{\Longrightarrow} c \underset{Bit_{\sqrt{N}-1}}{\Longrightarrow} \dots \ c \underset{Bit_{0}}{\Longrightarrow}$



2.10.1 LEACMD__BITREVERSECOMPLEXEVEN (cBREVEsf), LEACMD__BITREVERSECOMPLEXODD (cBREVOsf)

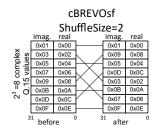
Complex vector X is re assorted applying bit reversed carry propagation addressing on 16-bit number pairs. This command performs an "In-Place" operation (output values are stored back to the location of the input values). The vector X must be address aligned to an address boundary with a modulo-value of zero: *mod(address, blocksize) = zero*.

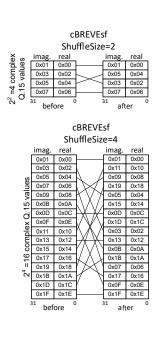
Arguments

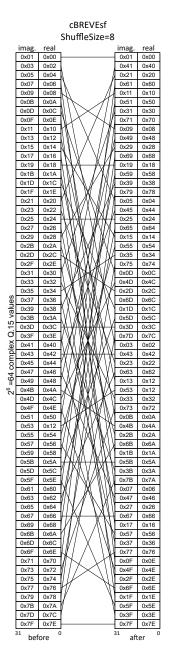
pointer to vector X as a 16-bit address value, pointing to first location of aligned vector X

shuffle size sqrt(N) a 16-bit unsigned value, determining how the values are interleaved (For the ...EVEN type this value is calculated taking the square root of the complex FFT points. For the ...ODD type this value is calculated by taking square root of the half or the complex FFT points.)

Examples









Uses

- Bit reversed sorting for FFT operations
- Time domain and frequency domain shuffling / de-shuffling avoiding continuous ordering to reduce the impacts on disturbances

Comments

Also see other shuffle sizes

cBREVOsf ShuffleSize=4 imag. real imag. real 0x01 0x00 0x01 0x00 0x03 0x02 0x21 0x20 0x05 0x04 0x11 0x10 0x07 0x06 0x31 0x30 0x08 0x09 0x08 0x09 0x0B 0x0A 0x29 0x28 0x18 0x0D 0x0C 0x19 0x0F 0x0E 0x39 0x38 0x11 0x10 0x05 0x04 0x13 0x12 0x25 0x24 values 0x15 0x14 0x15 0x14 0x17 0x16 0x35 0x34 0x19 0x18 0x0D 0x0C Q.15 0x1B 0x1A 0x2D 0x2C 0x1D 0x1C 0x1D 0x1C complex 0x1F 0x1E 0x3D 0x3C 0x21 0x20 0x03 0x02 0x23 0x22 0x23 0x22 0x25 0x24 0x13 0x12 =32 0x27 0x26 0x33 0x32 0x28 Š 0x29 0x0B 0x0A 0x2B 0x2A 0x2B 0x2A 0x2D 0x2C 0x1B 0x1A 0x2F 0x2E 0x3B 0x3A 0x31 0x30 0x07 0x06 0x33 0x32 0x27 0x26 0x35 0x34 0x17 0x16 0x37 0x36 0x37 0x36 0x39 0x0F 0x0E 0x38 0x3B 0x3A 0x2F 0x2E 0x3D 0x3C 0x1F 0x1E 0x3F 0x3E 0x3F 0x3E 31 31 0 0 before after

2.10.2 LEACMD___BITREVERSECOMPLEXLONGEVEN (cBREVEIf), LEACMD___BITREVERSECOMPLEXLONGODD (cBREVOIf)

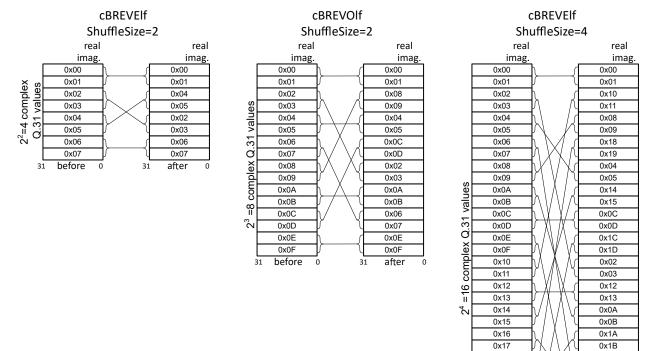
Complex vector X is re assorted applying bit reversed carry propagation addressing on 32 bit number pairs. This command in performs an "In-Place" operation (output values are stored back to the location of the input values). The vector X must be address aligned to an address boundary with a modulo-value of zero: *mod(address, blocksize) = zero*.

Arguments

pointer to vector X as a 16-bit address value, pointing to first location of aligned vector X

shuffle size sqrt(N) a 16-bit unsigned value, determining how the values are interleaved (For the ...EVEN type this value is calculated taking the square root of the complex FFT points. For the ...ODD type this value is calculated by taking square root of the half or the complex FFT points.)

Examples



Uses

- Bit reversed sorting for FFT operations
- Time domain and frequency domain shuffling / de-shuffling avoiding continuous ordering to reduce the impacts on disturbances

Comments

• Also see other shuffle sizes

0x06

0x07

0x16

0x17

0x0E

0x0F

0x1E 0x1F

after

31

0x18

0x19

0x1A

0x1B

0x1C

0x1D

0x1E

0x1F before

0

31

Texas

NSTRUMENTS

www.ti.com

2.11 Group 11: Post Operation for Real FFTs

Table 2-11 lists the parameter structure for this group.

				input1	reserved1	vectorSize	log2Size
				V _{s0} *	occu.	Siz	
				=& X	ŏ	=N	lb(N)
LEACMD_	_FFT	(FFTSPLIT)	0x28	Q.15		u16	u16
LEACMD_	_FFTLONG	(FFTLONGSPLIT)	0xA4	Q.31		u16	u16

Table 2-11. Group 11 Parameters: Post Operation for Real FFTs

2.11.1 LEACMD__FFTSPLIT (SPLIFsf/rFFTsf), LEACMD__FFTLONGSPLIT (SPLITIf/rFFTIf)

Complex vector X is converted from interleaved complex FFT results to positive spectrum only complex FFT results by applying a modified Edson's algorithm. This command in performs an "In-Place" operation (output values are stored back to the location of the input values). The vector X must be address aligned to an address boundary with a modulo-value of zero: *mod(address, blocksize) = zero*.

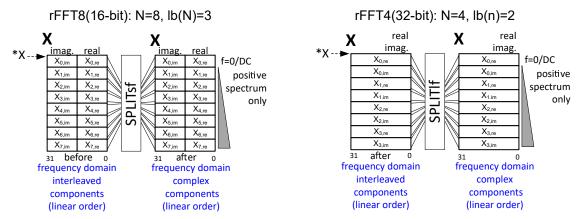
Arguments

pointer to vector X as a 16-bit address value, pointing to first location of aligned vector X

vector size (N) a 16-bit unsigned value, defining number of real points (for rFFT16 set to 16, for rFFT128 set to 128, and so on)

log size lb(N) a 16-bit unsigned number, defining the interleave factor (for rFFT 16 set to 4, for rFFT128 set to 7, and so on)

Example



Uses

• Post processing on complex FFTs to allow real values input samples

Comments

• Be aware of the higher FFT gain (+6 dB) when using these commands

2.12 Group 12: De-Interleave and Sort Functions

Table 2-12 lists the parameter structure for this group.

		=** * *	occu. reserved1	= <u>©</u> vectorSize	∎ InterleaveDepth	= % output	reserved2	
LEACMDDEINTERLEAVEEVENEVEN	0x58	IQ16		u16	u16	IQ16	rsvd.	
LEACMDDEINTERLEAVEEVENODD	0x5C	IQ16		u16	u16	IQ16	rsvd.	
LEACMDDEINTERLEAVEODDEVEN	0x60	IQ16		u16	u16	IQ16	rsvd.	$z_n = x_{n*M}$
LEACMDDEINTERLEAVEODDODD	0x64	IQ16		u16	u16	IQ16	rsvd.	n=0N-1
LEACMDDEINTERLEAVELONG	0x12C	IQ32		u16	u16	IQ32	rsvd.	

Table 2-12. Group 12 Parameters: De-Interleave and Sort Functions



2.12.1 LEACMD__DEINTERLEAVEEVENEVEN (rSPLITEEs), LEACMD__DEINTERLEAVEEVENODD (rSPLITEOs), LEACMD__DEINTERLEAVEODDEVEN (rSPLITOEs) LEACMD__DEINTERLEAVEODDODD (rSPLITOOs), LEACMD__DEINTERLEAVELONG (rSPLITI)

These commands are used for vector rearrangement of 16-bit and 32-bit values. By copying elements from the X source vector to the destination vector Z.

Mathematical notation

 $Z_n = X_{n^*M}$ with n = 0 to (N-1)

On 16-bit variants the first EVEN/ODD selector defines the 16-bit start location, the second EVEN/ODD is determines by the interleave factor M (if even/odd).

Arguments

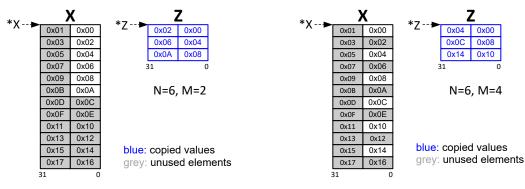
pointer to vector X as a 16-bit address value, pointing to first location of vector X (32 bit aligned)

pointer to vector Y as a 16-bit address value, pointing to first location of vector Z (32 bit aligned)

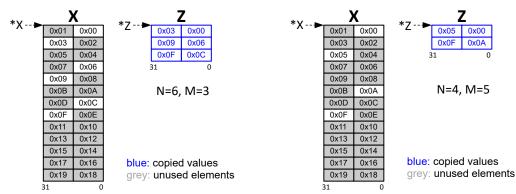
interleave depth M as a 16-bit value determining the de-interleave pattern

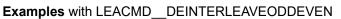
vector size N as a 16-bit unsigned value, defining the count of copy operations

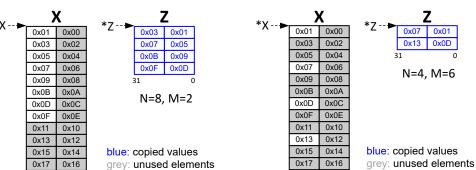
Examples with LEACMD__DEINTERLEAVEEVENEVEN



Examples with LEACMD__DEINTERLEAVEEVENODD



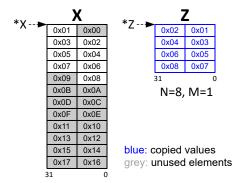


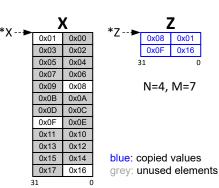


31

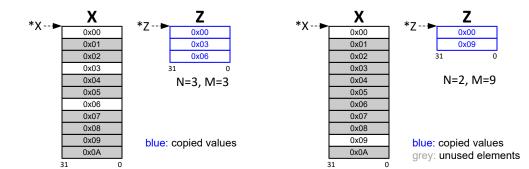
Examples with LEACMD__DEINTERLEAVEODDODD

31





Examples with LEACMD__DEINTERLEAVELONG



Uses

- Split 32-bit vector into 16-bit vectors
- · Split complex vectors into real- and imaginary sub-vectors
- Rearrange SIMD vector pairs
- · Rearrange matrices
- Rotate matrices orthogonal
- Swap real with imaginary in16-bit vector pairs
- · Filling vectors

Comments

• Also see irregular pattern for additional uses

EXAS

RUMENTS

www.ti.com

Common de-interleave pattern

				16	B destina	tion vecto	ion vector (index and odd/even)							
int	terleaveD	epth	0 E	1 0	2 E	3 0	4 E	5 🔾	6 E	7 0	-			
		-3 -4	0100	0000	00F8	0000	00F0	0000	00E8	0000				
	z	-1 -2	0100	0000	00FC	0000	00F8	0000	00F4	0000				
	EVENEVEN	0 (1)	0100	0100	0100	0100	0100	0100	0100	0100	filleven			
	VEN	2 (3)	0100	0102	0104	0106	0108	010A	010C	010E	-			
	ш Н	4 (5)	0100	0104	0108	010C	0110	0114	0118	011C				
		6 (7)	0100	0106	010C	0112	0118	011E	0124	012A				
		-4 -5	0100	0000	00F6	0000	00EC	0000	00E2	0000	1			
	0	-2 -3	0100	0000	00FA	0000	00F4	0000	00EE	0000				
:	EVENODD	-1 (0)	0100	0000	00FE	0000	00FC	0000	00FA	0000				
VE.		1 (2)	0100	0101	0102	0103	0104	0105	0106	0107	сору			
LEA		3 (4)	0100	0103	0106	0109	010C	010F	0112	0115				
LER		5 (6)	0100	0105	010A	010F	0114	0119	011E	0123				
		-3 -4	0101	0000	00F9	0000	00F1	0000	00E9	0000	1			
	z	-1 -2	0101	0000	00FD	0000	00F9	0000	00F5	0000				
ШШ	EVE	0 (1)	0101	0101	0101	0101	0101	0101	0101	0101	fillodd			
LEACMDDEINTERLEAVE	ODDEVEN	2 (3)	0101	0103	0105	0107	0109	010B	010D	010F				
1 1		4 (5)	0101	0105	0109	010D	0111	0115	0119	011D				
		6 (7)	0101	0107	010D	0113	0119	011F	0125	012B				
		-4 (-5)	0101	0000	00F7	0000	00ED	0000	00E3	0000				
		-2 (-3)	0101	0000	00FB	0000	00F5	0000	00EF	0000				
	0	0 (-1)	0101	0000	00FF	0000	00FD	0000	00FB	0000				
	адоддо	1 (2)	0101	0102	0103	0104	0105	0106	0107	0108	16B_shift			
	ō	3 (4)	0101	0104	0107	010A	010D	0110	0113	0116				
		5 (6)	0101	0106	010B	0110	0115	011A	011F	0124				
		7 (8)	0101	0108	010F	0116	011D	0124	012B	0132				
even lo	cation in	16B source	evector			Detterm								
odd loc	ation in 1	6B source	vector				rom incren at value 0x		d source ve	ector				
irregula	r pattern													
	NG	-3	100	FD	FA	F7	F4	F1	EE	EB]			
	Ň	-2	100	FE	FC	FA	F8	F6	F4	F2				
DE.	VEL	-1	100	FF	FE	FD	FC	FB	FA	F9	rev_copy			
П	EA	0	100	100	100	100	100	100	100	100	fill			
LEACMD	ERI	1	100	101	102	103	104	105	106	107	сору			
Ľ	INTERLEAVELO	2	100	102	104	106	108	10A	10C	10E				
	:	3	100	103	106	109	10C	10F	112	115				



2.13 Group A: Programming Structure Functions

Table 2-13 lists the parameter structure for this group.

Table 2-13. Group A Parameters: Programming Structure Functions

		input	vectorSize
		=&L	N
LEACMD_MOVELONGLIST C	x80	u32	u16
LEACMD_SUSPEND 0	x00		
LEACMD_RESUME 0	x08		
LEACMD_RESUME 0	x08		

2.13.1 LEACMD_MOVELONGLIST (LSTMOVI)

This command is used to rearrange variables in LEA accessible memory. A control list L contains all required information for this copy/move operation.

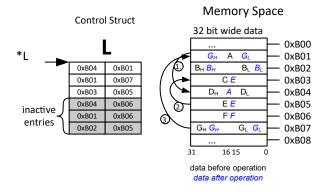
Arguments

pointer to list L as a 16-bit address value, pointing to first location of vector X (32 bit aligned)

vectorSize N as a 16-bit unsigned value, defining the number of 32-bit copy operations

control list as dual 16 bit address pairs with the source address in the lower 16 bits and destination address in the upper 16 bits or an 32-bit word.

Example for N = 3





2.13.2 LEACMD_SUSPEND (SUSPEND)

This command is used to save LEA internal registers to memory and stop operation.

Arguments

none

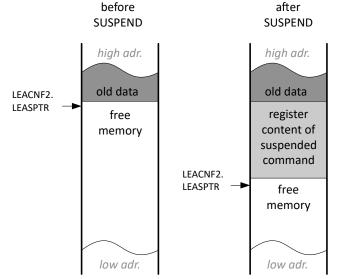


Table 2-14. LEA Sta	ack Frame Structure
Memory Address	Register Content Stored
LEASPTR + 100	LEADSTSTAT
LEASPTR + 96	LEACNF0
LEASPTR + 92	LEACNF1
LEASPTR + 88	LEACMDSTAT
LEASPTR + 84	LEAS1STAT
LEASPTR + 80	LEAS0STAT
LEASPTR + 76	PC-Saved
LEASPTR + 72	LoopEnd1 : LoopEnd0
LEASPTR + 68	LoopStart1 : LoopStart0
LEASPTR + 64	LoopCount1 : LoopCount0
LEASPTR + 60	Loop Index
LEASPTR + 56	Address Registers [SCL:SOV]
LEASPTR + 52	Address Registers [SR1:SR]
LEASPTR + 48	Address Registers [MR:PR]
LEASPTR + 44	Address Registers [PA3:PA2]
LEASPTR + 40	Address Registers [PA1:PA0]
LEASPTR + 36	Address Registers [MA:SA2]
LEASPTR + 32	Address Registers [SA1:SA0]
LEASPTR + 28	Data path register C7
LEASPTR + 24	Data path register C6
LEASPTR + 20	Data path register C5
LEASPTR + 16	Data path register C4
LEASPTR + 12	Data path register C3
LEASPTR + 8	Data path register C2
LEASPTR + 4	Data path register C1
LEASPTR + 0	Data path register C0

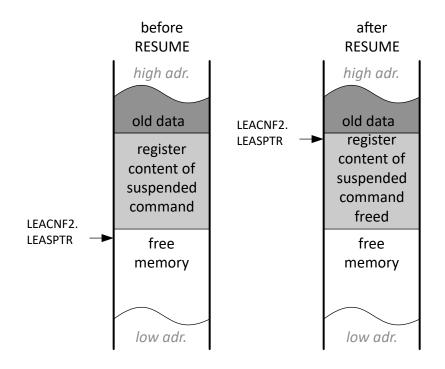


2.13.3 LEACMD__RESUME (RESUME)

This command is used to restore LEA internal registers from memory and resume operation.

Arguments

none



2.14 Group B: Special Functions for Math and Signal Processing

Table 2-15 lists the parameter structure for this group.

Table 2-15. Group B Parameters: Special Functions for Math and Signal Processing

LEACMDMPYMATRIXROW N×M	input1; & N _{ROW} of Q.15	
	reserved 1	$[n_0 \ n_1 \ \dots \ n_{R-1}] X$
	rowSize; R as u16 (rows of	$\begin{bmatrix} m_{0,0} & m_{0,1} & m_{0,2} & m_{0,C-1} \end{bmatrix}$
	colSize; C u16 (cols c	$ \begin{array}{c ccccccccccccccccccccccccccccccccccc$
	colVector; &M _{COL} of Q.15	$\begin{bmatrix} m_{R-1,0} & m_{R-1,1} & m_{R-1,2} & m_{R-1,C-1} \end{bmatrix}$
	0x34 output &Z _{OUT} of Q.15	
LEACMD_POLYNOMIALSCALAR	input1; X as Q.31	
	coeffs ;& C of Q.31	$- LEAPMDST = \sum_{m=1}^{\infty} (C_m * X_{\square}^m) * SF;$
	order ;M	$\sum_{m=0}^{m=0} (O_m + X_{\perp}) + O_{\perp},$
	0xBC scaleFactor ; SF as signed	
LEACMD_IIRBQ1	input1; &X of Q.15	$X_{n} \{-1,1\} \qquad \qquad A_{0} \{-1,1\} \qquad \qquad Q_{0} \{-1,1\} \qquad \qquad Z_{n} \{-1,1\} \qquad \qquad$
Direct form I structure	reserved 1	$ \begin{array}{ c c c c c c c c c c c c c c c c c c c$
	vectorSize; N of u16	Z ⁻¹ 0.5*B ₁ (-11)/// 0.5*A ₁ (-11) Z ⁻¹ S → Q1[Q.15] Q2[Q.15]
	output; &Z of Q.15	$\begin{array}{c c} Q_1 & Z & 0.0 & \text{S}(1, 10.1) \\ \hline Q_1 & Z & 0.0 & \text{S}(1, 10.1) \\ \hline & & & & & & & \\ \hline & & & & & & \\ \hline & & & &$
	state; & S of Q.15	
	coeffs; & C of Q.15	7-1 0.5*B ₁ {-11} 0.5*A ₁ {-11} 7-1 - B0[Q.15]
	direction; S16	$\begin{pmatrix} 1 & 1 \\ 1 $
	0x4C reserved	
LEACMD_IIRBQ2	input1; & X of Q.15	Bias MAX (X ₀) →→ MAX (X _E) Extended Functions
Direct form II structure	reserved 1	$(-11) \qquad \qquad$
	vectorSize; N ofu16	
	output; & Z of Q.15	$\underbrace{X_{n}(\cdot11)}_{A_{0}(\cdot11)} \bullet \underbrace{\Phi_{0}(\cdot11)}_{A_{0}(\cdot11)} \bullet \underbrace{\Phi_{0}(\cdot11$
	state; &S of Q 15	
	0x11C coeffs; &C of Q.15	$\begin{bmatrix} \bullet & \bullet \\ A_1 \{ 1,, 1 \} \end{bmatrix} = \begin{bmatrix} \bullet & \bullet \\ B_1 \{ -1,, 1 \} \end{bmatrix} = \begin{bmatrix} \bullet & \bullet \\ B_1 \{ -1,, 1 \} \end{bmatrix} = \begin{bmatrix} \bullet & \bullet \\ B_1 \{ -1,, 1 \} \end{bmatrix} = \begin{bmatrix} \bullet & \bullet \\ B_1 \{ -1,, 1 \} \end{bmatrix}$
LEACMD IIRBQ2EXTENDED	input1; &X of Q.15	
Direct form II structure	reserved 1	
	vectorSize; N ofu16	
	output; & Z of Q.15	$\begin{array}{c c c c c c c c c c c c c c c c c c c $
	state; &ES of Q.15	A0[Q.15] A1[Q1.14] B1[Q1.14] B0[Q.15] A1[Q1.14] B1[Q1.14] B0[Q.15]
	0x90 coeffs; &EC of Q.15	- B2[Q.15]
LEACMDWINDOW	input1; & X of Q.15	← N→
	reserved 1	for n=0K-1
	vectorSize; N of u16	$x_{N-n} = x_{N-n} * c_n \qquad \qquad \mathbf{x}$
	tapSize; K of u16	and $x_n = x_n * c_n$
	0x130 coeffs; & C of Q.15	



2.14.1 LEACMD__MPYMATRIXROW

This command is used for matrix multiplication of real valued 16-bit numbers. It calculates a whole row of the result matrix in one run. For a complete matrix multiplication this function needs to be applied as many times as X and Z has rows.

Mathematical representation

Arguments

pointer to matrix X as a 16-bit address value, pointing to first location of actual row vector of matrix X

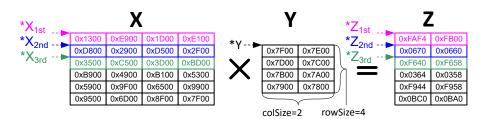
pointer to matrix Y as a 16-bit address value, pointing to first location of matrix X

pointer to matrix Z as a 16-bit address value, pointing to first location of actual row vector of matrix Z

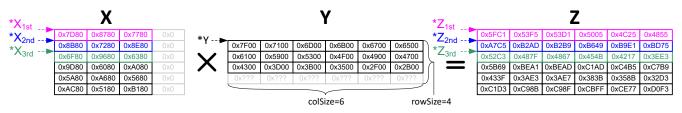
row size (rowSize) as a 16-bit number, defining the count of rows of Y (this number must be a multiple of two, pad with zeroes for odd row numbers)

column size (colSize) as a 16-bit number, defining the count of columns of Y (this number must be a multiple of two, pad with zeroes for odd row numbers)

Example 1



Example 2 original 6*3 matrix, extended to 6*4, row 3 of Y may be occupied with any data





2.14.2 LEACMD_POLYNOMIALSCALAR

Polynomial result of vector X and Vector C are written to vector register LEAPMDST using a 32-bit signed fractional multiplication of powers of X and pointwise values of C followed by a 32-bit accumulation and a scaling multiplication.

Mathematical notation

$$\sim PMDST = \left(\sum_{m=0}^{M-1} (C_m \cdot x_n^m)\right) \cdot SF \qquad \text{with } n = 0 \text{ to (N-1) and } m=0 \text{ to (M-1)}$$
$$LEAPMDST = \left(c_0 + c_1 \cdot x_n + c_2 \cdot x_n^2 + c_3 \cdot x_n^3 + c_4 \cdot x_n^4 \dots\right) \cdot SF \qquad \text{with } n = 0 \text{ to (N-1)}$$

Arguments

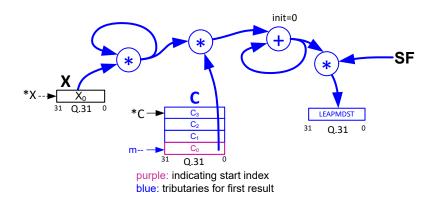
pointer to vector X as a 16-bit address value, pointing at an element of vector X

pointer to vector C as a 16-bit address value, pointing to the element of vector C with highest index (this vector is reverse ordered)

order M as a 16-bit unsigned value, defining the maximum order of the powers

scaling factor SF as 32-bit signed value used for post scaling

Example for M = 4



Uses

- Mathematical functions based on Taylor/McLaurin series
- · Approximation of nonlinear function using polynomial coefficient fitting methods
- Sensor linearization

Comments

- Some real applications problems require transposition and scaling of the polynomial coefficients to stay within its convergence range
- · Coefficient sets for some common functions are provide within the LEA coefficient ROM

2.14.3 LEACMD_IIRBQ1 (rBQ1sf)

This command represents a signal processing element of a second order biquadratic filter of "Direct Form 1". Samples from the input vector X are processed and stored in vector Z. The filter state S is picked up and stored back to/from a separate structure. This allows a continuation of operation for sample streams. The filter coefficients for A1, A2, B0, B1 and B2 are derived from those given in many text books (a1, a2, b0, b1, b2). It allows A2 to reach full -1 without epsilon offset. Scaling of the input signal is achieved by scaling down B0, B1, B2 with the same factor. A1 and B1 range from -2 to 2–2eps.

Mathematical notation of addressed function

$$H(z) = \frac{b0 + b1 * z^{-1} + b2 * z^{-2}}{1 + a1 * z^{-1} + a2 * z^{-2}}$$

Implemented function

$$H(z) = \frac{B0 + B1 * z^{-1} + B2 * z^{-2}}{1 - A1 * z^{-1} - A2 * z^{-2}}$$

Arguments

pointer to vector X as a 16-bit address value, pointing to first location of vector X (32 bit aligned)

pointer to vector Z as a 16-bit address value, pointing to first location of vector Z (32 bit aligned)

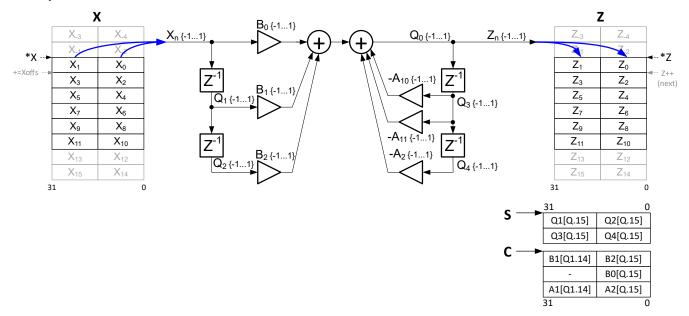
states pointer S a structure containing 16-bit values holding the states of the BQ filter element

coefficients pointer C a structure of 16-bit values holding the coefficients of the BQ filter element

vector size N as a 16-bit unsigned value, defining number of value pairs processed (N = 16 will calculate 32 samples of Q.15)

direction Xoffs as signed 16-bit value. This allowing the input pointer to increment or decrement by bigger offsets (Xoffs = 2 increments takes samples with the index order 0, 1, 4, 5, 8, 9, …).

Example for N = 6, Xoffs = 1



Note: The values of A1 and B1 reach up to twice the value of A2 and B2. For this reason The Term with A1 and B1 is added twice. This also can be seen that the value for A1 and B1 is interpreted as Q1.14 format, Also true is that A1 and B1 holds half the value in Q.15 format.

Multiple paradigms on this topic are addressed in common DSP literature.



2.14.4 LEACMD_IIRBQ2 (rBQ2sf)

This command represents a signal processing element of a second order biquadratic filter of "Direct Form 2" also known as canonic form. Samples from the Input vector X are processed and stored in vector Z. The filter's state S is picked up and stored back to/from a separate structure. This allows a continuation of operation for sample streams. The filter coefficients for A0, A1, A2, B0, B1 and B2 are derived from those given in many text books (a0, a1, a2, b0, b1, b2). A simple scaling step allows to simplify hardware while maintaining high accuracy. It allows A0 to be smaller than one and A2 to reach full -1 without epsilon offset. A1 and B1 range from -2 to 2–2eps.

Mathematical notation of addressed function

$$H(z) = a0 \quad \frac{b0 + b1 * z^{-1} + b2 * z^{-2}}{1 + a1 * z^{-1} + a2 * z^{-2}}$$

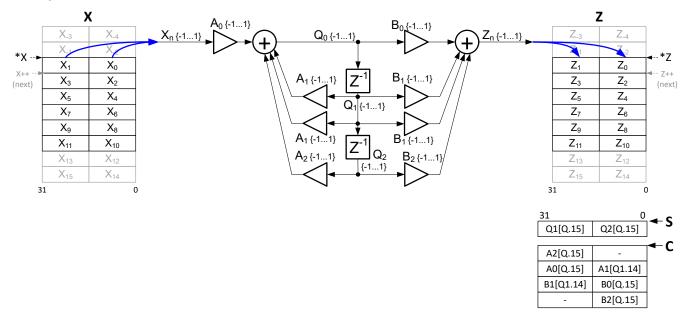
Implemented function

$$H(z) = A0 \frac{B0 + B1 * z^{-1} + B2 * z^{-2}}{1 - A1 * z^{-1} - A2 * z^{-2}}$$

Arguments

pointer to vector X as a 16-bit address value, pointing to first location of vector X (32 bit aligned) pointer to vector Z as a 16-bit address value, pointing to first location of vector Z (32 bit aligned) states pointer S a structure containing 16-bit values holding the states of the BQ filter element coefficients pointer C a structure of 16-bit values holding the coefficients of the BQ filter element vector size N as a 16-bit unsigned value, defining number of value pairs processed (N = 16 will calculate 32 samples of Q.15)

Example for N = 6



Note: The values of A1 and B1 reach up to twice the value of A2 and B2. For this reason, the term with A1 and B1 is added twice. This also can be seen that the value for A1 and B1 is interpreted as Q1.14 format. A1 and B1 hold half the value in Q.15 format.

Multiple paradigms on this topic are addressed in common DSP literature.

2.14.5 LEACMD_IIRBQ2EXTENDED (rBQ2Xsf)

This command represents a signal processing element of a second order biquadratic filter of "Direct Form 2" also known as canonic form. Samples from the Input vector X are processed and stored in vector Z. The filter state ES is picked up and stored back to/from a separate structure. This allows a continuation of operation for sample streams. The filter coefficients for A0, A1, A2, B0, B1 and B2 are derived from those given in many text books (a0, a1, a2, b0, b1, b2). A simple scaling step allows to simplify while maintaining high accuracy. It allows A0 to be smaller than one and A2 to reach -1 without epsilon offset. A1 and B1 range from -2 to 2–2eps.

Mathematical notation of addressed function

$$H(z) = a0 \quad \frac{b0 + b1 * z^{-1} + b2 * z^{-2}}{1 + a1 * z^{-1} + a2 * z^{-2}}$$

Implemented function

$$H(z) = Bias + A0 \frac{B0 + B1 * z^{-1} + B2 * z^{-2}}{1 - A1 * z^{-1} - A2 * z^{-2}}$$

Arguments

pointer to vector X as a 16-bit address value, pointing to first location of vector X (32 bit aligned)

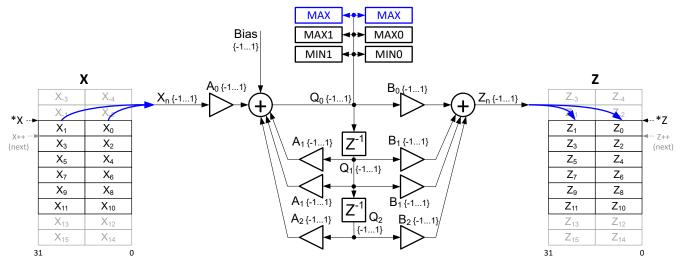
pointer to vector Z as a 16-bit address value, pointing to first location of vector Z (32 bit aligned)

states pointer ES a structure containing 16-bit values holding the states, minima and maxima of the extended BQ filter element

coefficients pointer C a structure of 16-bit values holding the coefficients of the extended BQ filter element

vector size N as a 16-bit unsigned value, defining number of value pairs processed (N = 16 will calculate 32 samples of Q.15)

Example for N = 6



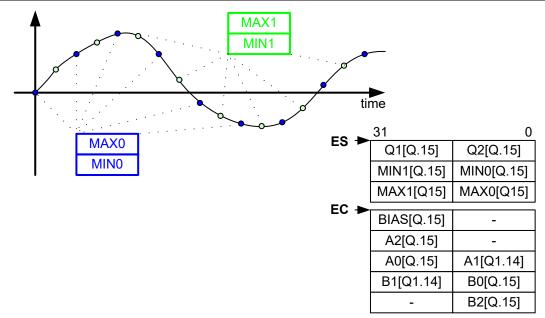


The minima and maxima of the odd indexed samples are stores in MAX1/MIN1

The minima and maxima of the all samples are kept in MIN/MAX vs.

To obtain the minima and maxima of all samples, calculate the min and max or the MIN0/MIN1 and MAX0/MAX1 using the CPU.





Note: The values of A1 and B1 reach up to twice the value of A2 and B2. For this reason, A1 and B1 is added twice. This also can be seen that the value for A1 and B1 is interpreted as Q1.14 format. A1 and B1 hold half the value in Q.15 format.

Multiple paradigms on this topic are addressed in common DSP literature.



2.14.6 LEACMD_WINDOW

This command performs a pointwise windowing operation on vector Z with the coefficient vector C that describes the window. C holds only half of the window coefficient as the command uses those coefficients also mirrored.

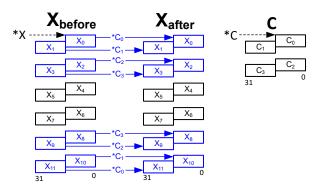
Mathematical notation

for
$$n = 0...K-1$$

 $x_{N-n} = x_{N-n} * c_n$
and $x_n = x_n * c_n$

Arguments

pointer to vector X as a 16-bit address value, pointing to first location of vector X (32 bit aligned) pointer to vector C as a 16-bit address value, pointing to first location of vector Z (32 bit aligned) tap size K as a 16-bit unsigned value defining length of coefficient vector C (K is a multiple of 2) vector size N as a 16-bit unsigned value, defining number of value pairs processed (N is a multiple of 2) **Example** for N = 12, K = 4



3 LEA Hardware

This chapter is a short overview of the LEA ROM and the other hardware-related features of the LEA module.

3.1 LEA ROM Constants

The LEA module features an internal ROM that holds coefficients for commonly used mathematical functions and auxiliary vectors that can be used to emulate advanced vector operations or as vector constants. Table 3-1 and Table 3-2 show the ROM content of the LEA module.

Name	e 3-1. LEA Common Functions Po	3	Address {LEA} Width		
LEA CONST LONG SQRT	coefficients for SQRT(X-1)	0x8000	32B	33	
LEA CONST LONG EXP	coefficients for EXP(X)	0x8021	32B	13	
LEA CONST LONG 2COS	coefficients for COS(X^2)	0x802E			
LEA_CONST_LONG_SIN	coefficients for SIN(X)	0x8035	32B	12	
LEA_CONST_LONG_COS	coefficients for COS(X)	0x8041	32B	13	
LEA_CONST_LONG_INV	coefficients for 1/X	0x804E	32B	33	
LEA_CONST_LONG_2SECH	coefficients for SECH(X^2)	0x806F	32B	25	
LEA_CONST_LONG_CBTF	coefficients for CBRT(X) cubic root	0x8088	32B	33	
LEA_CONST_SQRTW	coefficients for SQRT(X/PI) wide range	0x80A1	16B	5	
LEA_CONST_SQRT	coefficients for SQRT(X-1)	0x80A4	16B	13	
LEA_CONST_SINW	coefficients for SIN(X/PI)	0x80AB	16B	7	
LEA_CONST_SIN	coefficients for SIN(X)	0x80AF	16B	8	
LEA_CONST_COS	coefficients for COS(X)	0x80B3	16B	9	
LEA_CONST_INV	coefficients for 1/X	0x80B8	16B	17	
LEA_CONST_TAN	coefficient for TAN(X)	0x80C1	32B	12	
LEA_CONST_LONG_ASIN	coefficient for ASIN(x)	0x80CD	32B	24	
LEA_CONST_LONG_ATN1	coefficient for ATN(x) segment 1	0x80E5	32B	6	
LEA_CONST_LONG_ATN2	coefficient for ATN(x) segment 2	0x80EB	32B	9	
LEA_CONST_LONG_ATN3	coefficient for ATN(x) segment 3	0x80F4	32B	10	
LEA_CONST_LONG_2EXP	coefficient for EXP(X^2)	0x80FE	32B	10	
LEA_CONST_LONG_LOG	coefficient for LOG(X)	0x8108	32B	13	
LEA_CONST_LONG_4EXP	coefficient for EXP(X) auxiliary	0x8115 32B		10	
LEA_CONST_LONG_2SIN	coefficient for SIN(X^2)	0x811F	32B	4	

Table 3-1. LEA Common Functions Polynomial Coefficients

Table 3-2. LEA Auxiliary Vectors

Name Typical Use	Value	Address Range {Lea}	Width	Size
LEA_CONST_0 neutral element, dummy source	0x0	0xA000 to 0xA3FF	16B (dual)	2048
LEA_CONST_1 positive epsilon	0x1	0xA400 to 0xA7FF	16B (dual)	2048
LEA_CONST_4 positive amplitude margin 16B, + 0.5	0x4000	0xA800 to 0xABFF	16B (dual)	2048
LEA_CONST_7 neutral element, +1	0x7FFF	0xAC00 to 0xAFFF	16B (dual)	2048
LEA_CONST_8 substitutions, complementary element, conversion of ADC/DAC formats, -1,	0x8000	0xB000 to 0xB3FF	16B (dual)	2048
LEA_CONST_C negative amplitude margin 16B,5	0xC000	0xB400 to 0xB7FF	16B (dual)	2048
LEA_CONST_E negative double epsilon, inverse element	0xFFFE	0xB800 to 0xBBFF	16B (dual)	2048

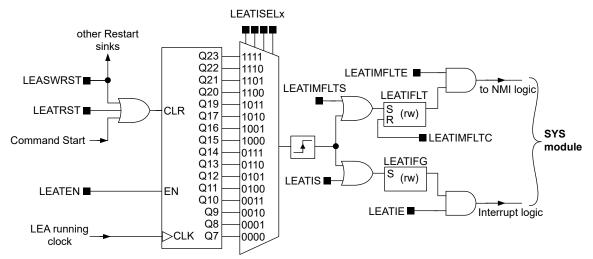


Table 3-2. LEA Auxiliary Vectors (continued)						
Name Typical Use	Value	Address Range {Lea} Width		Size		
LEA_CONST_F negative epsilon	0xFFFF	0xBC00 to 0xBFFF	16B (dual)	2048		
LEA_CONST_LONG_0 neutral element, dummy source	0x0	0xC000 to 0xC7FF	32B	2048		
LEA_CONST_LONG_1 positive epsilon			32B	2048		
LEA_CONST_LONG_4 positive amplitude margin 32B, +.5	0x4000:0000	0xD000 to 0xD7FF	32B	2048		
LEA_CONST_LONG_7 neutral element	0x7FFF:FFFF	0xD800 to 0xDFFF	32B	2048		
LEA_CONST_LONG_8 substitutions, complementary element, conversion of ADC/DAC formats,	0x8000:0000	0xE000 to 0xE7FF	32B	2048		
LEA_CONST_LONG_C negative amplitude margin 32B,5	0xC000:0000	0xE800 to 0xEFFF 32B		2048		
LEA_CONST_LONG_E negative double epsilon, inverse element	0xFFFF:FFFE	0xF000 to 0xF7FF 32B		2048		
LEA_CONST_LONG_F negative epsilon	0xFFFF:FFFF	0xF800 to 0xFFFF	32B	2048		

Table 3-2. LEA Auxiliary Vectors (continued)

3.2 LEA Timer

The LEA module features a simple timeout timer that can be used for LEA oriented applications (for example, for task alignment or as a simple command timeout timer). The timer is running with the LEA clock and is counting up. It is reset each time a new LEA command is started. This timer can trigger interrupts and NMI by enabling the corresponding enable flags.





3.3 LEA and Debug

The LEA module is a module that accelerates complex mathematical functions. It can be compared with the hardware multiplier that accelerates multiplications or the DMA controller that accelerates data transfers. LEA has no interface to CPU's debugging module. Debugging of LEA applications is done by using The CPU's debugging features and memory access features of the debugger. LEA may execute code while the CPU is in LPMx in some operation modes. Therefore, breakpoint may not always be possible directly after the command invoke instruction. This is caused due to the fact that the CPU is put into an artificial LPM like state when hitting a breakpoint or during single step. In those phases, LEA may still execute. Inserting a NOP instructions after the command invoke instruction allows to set breakpoints as usual.

3.4 LEA Hardware Registers

Figure 3-2 shows an overview of the most important hardware registers of the LEA.

LEACNF0	31/15 ~TISEL	~TEN ~T	RST ~MEMFLTS	~CFLTS	~TIMFL	.TS	~/	REES	16/ ~DONES
L	~MEMFLTE	~TIMFLT	e ~ilb	~ILPM	~L	PR		FLTHOLD	~SWRS
н	31/15 LEAWRSTAT	LEARDSTAT	~MEMFLTC	~CFLTC	~TIMFLTC		~	REEC	16, ~DONEC
LEACNF1	LEAASST		LEAPWST		LEAMODE				~BUSY
	31/15						· · · · · · · · · · · · · · · · · · ·		16
LEACNF2							1		
L	31/15			LEASPTR			1		16
H LEACMA			LEAC	MDP (high wor	d)		1		
LENTONINT			LEAC	MDP (low word	d)				
Н	31/15	1		LEAAP			1		16
LEACMCTL L					LEA	CROFF	~DEC ~ING		~MA
Н	31/15						1		16
ACMDSTAT		1	_	LEACN	MD (working col	mmand)	1	~	ITFLG
L	31/15						1		16
LEASOSTAT H			LEASOVAL / LEAS1V						
LEAS1STAT L	24/45		LEASOVAL / LEAS1V	AL (low word ;)	working register)			
eadststat h	31/15	LEAD	STVAL / LEAPMDST (hig	l h word ; worki	ng result / result	t register)			16
LEAPMDST L		LEAD	STVAL / LEAPMDST (lov	v word ; workir	ng buffer / result	register)			
н	31/15						1		16
LEAPMCTL				~TRG					~CMDE
	31/15	1					1		. 16
LEAPMSO H			LEAPMSO / LEAPM				1		
LEAPMS1 L			LEAPMS0 / LEAPM	SI (IOW WORD ;	source register)		1		16
H LEAPMCB		LE4	CTX (command context)					
LEATINED				LEAC	CMD (command	buffer)		~	ITFLG
н	31/15	1				l			16
<i>LEAIFGSET</i> L					~CMDIS	~SDIIS	~OORIS	~TIS	~OVLIS
н	31/15	I				L			16
LEAIE		I	I		~CMDIE	~SDIIE	~OORIE	~TIE	~OVLIE
-									16
H LEAIFG							1 1		
L	.21/15				~CMDIFG	~SDIIFG	~OORIFG	~TIFG	~OVLIF
	31/15								16
H LEAIV		1	1			1			





4 Revision History

NOTE: Page numbers for previous revisions may differ from page numbers in the current version.

DATE	REVISION	NOTES
March 2021	*	Initial Release

IMPORTANT NOTICE AND DISCLAIMER

TI PROVIDES TECHNICAL AND RELIABILITY DATA (INCLUDING DATA SHEETS), DESIGN RESOURCES (INCLUDING REFERENCE DESIGNS), APPLICATION OR OTHER DESIGN ADVICE, WEB TOOLS, SAFETY INFORMATION, AND OTHER RESOURCES "AS IS" AND WITH ALL FAULTS, AND DISCLAIMS ALL WARRANTIES, EXPRESS AND IMPLIED, INCLUDING WITHOUT LIMITATION ANY IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR NON-INFRINGEMENT OF THIRD PARTY INTELLECTUAL PROPERTY RIGHTS.

These resources are intended for skilled developers designing with TI products. You are solely responsible for (1) selecting the appropriate TI products for your application, (2) designing, validating and testing your application, and (3) ensuring your application meets applicable standards, and any other safety, security, regulatory or other requirements.

These resources are subject to change without notice. TI grants you permission to use these resources only for development of an application that uses the TI products described in the resource. Other reproduction and display of these resources is prohibited. No license is granted to any other TI intellectual property right or to any third party intellectual property right. TI disclaims responsibility for, and you will fully indemnify TI and its representatives against, any claims, damages, costs, losses, and liabilities arising out of your use of these resources.

TI's products are provided subject to TI's Terms of Sale or other applicable terms available either on ti.com or provided in conjunction with such TI products. TI's provision of these resources does not expand or otherwise alter TI's applicable warranties or warranty disclaimers for TI products.

TI objects to and rejects any additional or different terms you may have proposed.

Mailing Address: Texas Instruments, Post Office Box 655303, Dallas, Texas 75265 Copyright © 2022, Texas Instruments Incorporated