## Reference Guide

## Low-Energy Accelerator (LEA) Commands

## 4if 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 .....  3
1.2 Invoking a Command ..... 4
1.2.1 Command Invoke Methods .....  6
1.2.2 Command Execution in Low-Power Modes .....  6
1.3 LEADONE and LEAFREE Indications ..... 7
1.3.1 LEA Done Indication ..... 7
1.3.2 LEAFREE Indication ..... 7
2 LEA Commands ..... 8
2.1 Group 1: Basic Pointwise Vector or Matrix Operations .....  8
2.1.1 LEACMD__ADDLONGMATRIX (rADDMI) .....  9
2.1.2 LEACMD ADDMATRIX (rADDMs) ..... 10
2.1.3 LEACMD__MPYMATRIX (rMPYMsf) ..... 11
2.1.4 LEACMD__MPYLONGMATRIX (rMPYMIf) ..... 12
2.1.5 LEACMD _ MPYCOMPLEXMATRIX (cMPYMsf) ..... 13
2.1.6 LEACMD _ SUBMATRIX (rSUBMs) ..... 14
2.1.7 LEACMD__SUBLONGMATRIX (rSUBI) ..... 15
2.2 Group 2: Basic Vector MAC Operations ..... 16
2.2.1 LEACMD MAC (rMACsf) ..... 16
2.2.2 LEACMD__MAC3 (rMAC3sf) ..... 17
2.2.3 LEACMD__SCALEDMAC (sMACsf) ..... 18
2.3 Group 3: MAC, Point-Wise FIR, Correlation, and Convolution Operations ..... 19
2.3.1 LEACMD_MACMATRIX (rMACMsf) ..... 20
2.3.2 LEACMD__MACLONGMATRIX (rMACMIf) ..... 21
2.3.3 LEACMD__MACCOMPLEXMATRIX (cMACMsf) ..... 22
2.3.4 LEACMD__MACCOMPLEXCONJUGATEMATRIX (cMACMCONJsf) ..... 23
2.4 Group 4: Basic Min/Max Vector Search on 16B Data ..... 24
2.4.1 LEACMD__MAX (rMAXs) ..... 24
2.4.2 LEACMD__MIN (rMINs) ..... 25
2.4.3 LEACMD_MAXUNSIGNED (rMAXus) ..... 26
2.4.4 LEACMD__MINUNSIGNED (rMINus) ..... 27
2.5 Group 5: Generic Min/Max Vector Search on 32B Data ..... 28
2.5.1 LEACMD__MAXLONGMATRIX (rMAXMI) ..... 28
2.5.2 LEACMD MINLONGMATRIX (rMINMI) ..... 29
2.5.3 LEACMD__MAXUNSIGNEDLONGMATRIX (rMAXMul) ..... 30
2.5.4 LEACMD__MINUNSIGNEDLONGMATRIX (rMINMul) ..... 31
2.6 Group 6: Basic Min/Max Vector Search on Dual 16B Data ..... 32
2.6.1 LEACMD__MAXMATRIX (rMAXMs) ..... 33
2.6.2 LEACMD__MINMATRIX (rMINMs). ..... 34
2.6.3 LEACMD__MAXUNSIGNEDMATRIX (rMAXMus) ..... 35
2.6.4 LEACMD__MINUNSIGNEDMATRIX (rMINMus) ..... 36
2.7 Group 7: Block FIR, Correlation, Convolution Functions ..... 37
2.7.1 LEACMD__FIR (rFIRsf) ..... 38
2.7.2 LEACMD__FIRLONG (rFIRIf). ..... 40
2.7.3 LEACMD__FIRCOMPLEX (cFIRsf). ..... 42
2.7.4 LEACMD__FIRCOMPLEXLONG (cFIRIf) ..... 44
2.7.5 LEACMD__SCALEDFIR (sFIRsf). ..... 46
2.8 Group 8: Taylor McLaurin Functions on Pointwise Vectors or Matrices ..... 47
2.8.1 LEACMD__POLYNOMIAL (rPOLYsf). ..... 48
2.8.2 LEACMD__POLYNOMIALLONG (rPOLYIf) ..... 49
2.9 Group 9: Fast Fourier Transforms and Inverses ..... 50
2.9.1 LEACMD__FFTCOMPLEXAUTOSCALING (cFFTsf_as) ..... 51
2.9.2 LEACMD__FFTCOMPLEXFIXSCALING (cFFTsf_fs) ..... 53
2.9.3 LEACMD__FFTCOMPLEXLONG (cFFTIf). ..... 54
2.10 Group 10: Bit Reversed Pre-Sort Functions for FFTs. ..... 55
2.10.1 LEACMD__BITREVERSECOMPLEXEVEN (cBREVEsf), LEACMD__BITREVERSECOMPLEXODD (cBREVOsf). ..... 56
2.10.2 LEACMD _ BITREVERSECOMPLEXLONGEVEN (cBREVEIf), LEACMD__BITREVERSECOMPLEXLONGODD (cBREVOIf) ..... 58
2.11 Group 11: Post Operation for Real FFTs ..... 59
2.11.1 LEACMD__FFTSPLIT (SPLIFsf/rFFTsf), LEACMD__FFTLONGSPLIT (SPLITIf/rFFTIf) ..... 59
2.12 Group 12: De-Interleave and Sort Functions ..... 60
2.12.1 LEACMD__DEINTERLEAVEEVENEVEN (rSPLITEEs),
LEACMD__DEINTERLEAVEEVENODD (rSPLITEOs), LEACMD__DEINTERLEAVEODDEVEN (rSPLITOEs) LEACMD__DEINTERLEAVEODDODD (rSPLITOOs), LEACMD__DEINTERLEAVELONG (rSPLITI). ..... 61
2.13 Group A: Programming Structure Functions ..... 64
2.13.1 LEACMD__MOVELONGLIST (LSTMOVI) ..... 64
2.13.2 LEACMD__SUSPEND (SUSPEND) ..... 65
2.13.3 LEACMD__RESUME (RESUME) ..... 66
2.14 Group B: Special Functions for Math and Signal Processing ..... 67
2.14.1 LEACMD__MPYMATRIXROW ..... 68
2.14.2 LEACMD__POLYNOMIALSCALAR ..... 69
2.14.3 LEACMD__IIRBQ1 (rBQ1sf) ..... 70
2.14.4 LEACMD__IIRBQ2 (rBQ2sf). ..... 71
2.14.5 LEACMD__IIRBQ2EXTENDED (rBQ2Xsf). ..... 72
2.14.6 LEACMD__WINDOW. ..... 74
3 LEA Hardware ..... 75
3.1 LEA ROM Constants. ..... 75
3.2 LEA Timer ..... 76
3.3 LEA and Debug. ..... 76
3.4 LEA Hardware Registers ..... 77
4 Revision History ..... 78

## Trademarks

All trademarks are the property of their respective owners.

InSTRUMENTS
www.ti.com

## 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.

Table 1-1. LEA Command Groups

| 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 |

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.

```
LEACNF0=0; LEACNF1=0; LEACNF2=LeaSP; //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 New\overline{LEA VAR(vār) 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 N=4
void main(void) {
    WDTCTL = WDTPW | WDTHOLD; // Stop watchdog timer
    LEACNFO=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 ~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 $\qquad$ MAC holds four short values for the base parameter and two additional short parameters for the extension.

TEXAS INSTRUMENTS


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 ( $0 x C C D$ ).

## 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; // address of input1 vector containing values of short
    short reservedi; // reserved; may be used for context
    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_macParams; // 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=LEACMD_ $x x x+0 ;$ | LEA command without any further indication | 0 |
| LEAPMCBL=LEACMD_ $x x x+1 ;$ | LEA command with explicit result update | 1 |
| LEAPMCBL=LEACMD_ $x x x+2 ;$ | LEA command with interrupt upon completion | 2 |
| LEAPMCBL=LEACMD_ $x x x+3 ;$ | LEA command with interrupt and explicit result update | 3 |

### 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.


Figure 1-3. LEA Operation Modes INSTRUMENTS
www.ti.com

### 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 $=1 \mid 3$
- 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 $=0 \mid 2$ 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 $=1 \mid 3$ 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.


## 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

|  |  |  |  |  |  | $\begin{array}{r} \begin{array}{r} \stackrel{3}{2} \\ \text { in } \\ \vdots \\ \hline V_{0}^{*} \\ =\& \mathbf{Z} \end{array} \end{array}$ |  |  |  |  |
| :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: |
| LEACMD_ADDMATRIX | 0x40 | IQ16 |  | u16 | IQ16 | IQ16 | s16 | s16 | s16 | $z_{n}=x_{n}+y_{n}$ |
| LEACMD_ADDLONGMATRIX | 0x7C | IQ32 |  | 416 | IQ32 | IQ32 | s16 | s16 | s16 | $\mathrm{n}=0 \ldots \mathrm{~N}-1$ |
| LEACMD__MPYMATRIX | $0 \times 3 \mathrm{C}$ | Q. 15 |  | 416 | Q. 15 | Q. 15 | s16 | s16 | s16 |  |
| LEACMD_MPYLONGMATRIX | 0x74 | Q. 31 |  | 416 | Q. 31 | Q. 31 | s16 | s16 | s16 | $z_{n}=x_{n} * y_{n} ;$ |
| LEACMD__MPYCOMPLEXMATRIX | 0x78 | cQ. 15 |  | 416 | cQ. 15 | CQ. 15 | s16 | s16 | s16 |  |
| LEACMD_SUBMATRIX | 0x70 | IQ16 |  | 416 | IQ16 | IQ16 | s16 | s16 | s16 |  |
| LEACMD__SUBLONGMATRIX | 0xD4 | IQ32 |  | 416 | IQ32 | IQ32 | s16 | s16 | s16 | $\mathrm{n}=0 \ldots \mathrm{~N}-1$ |

### 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 $0 \times 7 F F F F F F F \equiv Q 31(1-\varepsilon)$ and $0 \times 80000000 \equiv$ Q31(-1).

## Mathematical notation

$\mathrm{z}_{\mathrm{n}}=\mathrm{x}_{\mathrm{n}}+\mathrm{y}_{\mathrm{n}} \quad$ with $\mathrm{n}=0$ to $(\mathrm{N}-1)$

## 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, ${ }^{*} \mathrm{Z}$ is incremented with after each operation vector size N as a 16-bit unsigned value; defining the number of operations performed

Example for $\mathrm{Xi}=\mathrm{Yi}=\mathrm{Zi}=1, \mathrm{~N}=8$
$\substack{1^{\text {t }} \\ 2^{\text {nd }} \\ 2^{\text {d }} \\ 3^{\text {rd }} \\ 4^{4 t} \\ 5^{\text {th }} \\ \vdots \\ \mathrm{N}-1^{\text {st }} \\ \mathrm{N}^{\text {th }}}$


## 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 $\mathrm{Xi}=0, \mathrm{Yi}=0, \mathrm{Zi}=1$. (for gapless fill)
- Address mirrored additions and copy using negative Xi , Yi or Zi
- Interleave or de-interleave and copy using other increment values.


## Comments

- 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 $0 \times 7 \mathrm{FFF}=\mathrm{Q} 15(1-\varepsilon)$ and $0 \times 8000 \equiv \mathrm{Q} 15(-1)$.

## Mathematical notation

$\mathrm{z}_{\mathrm{n}}=\mathrm{x}_{\mathrm{n}}+\mathrm{y}_{\mathrm{n}}$ with $\mathrm{n}=0$ to ( $\mathrm{N}-1$ ) \{ 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 $\mathrm{Xi}=\mathrm{Yi}=\mathrm{Zi}=2, \mathrm{~N}=16$


## 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 $\mathrm{Xi}=0, \mathrm{Yi}=0, \mathrm{Zi}=2$. (for gapless fill)
- Upscaling of vectors (for example, $X+X=2 X$ )
- Address mirrored additions and copy using negative $\mathrm{Xi}, \mathrm{Yi}$ or Zi
- Interleave or de-interleave and copy using other increment values


## Comments

- 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

$\mathrm{z}_{\mathrm{n}}=\mathrm{x}_{\mathrm{n}} \cdot \mathrm{y}_{\mathrm{n}}$ with $\mathrm{n}=0$ to $(\mathrm{N}-1)\{\mathrm{N}$ : 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 $X i$ 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 $\mathrm{Xi}=\mathrm{Yi}=\mathrm{Zi}=2, \mathrm{~N}=16$


## Uses

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


## Comments

- 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 $X i$ 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, ${ }^{*} \mathrm{Z}$ is incremented with after each operation vector size N as a 16-bit unsigned value; defining the number of operations performed

Example for $\mathrm{Xi}=\mathrm{Yi}=\mathrm{Zi}=1, \mathrm{~N}=8$


## Uses

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


## Comments

- 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. INSTRUMENTS


### 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 $0 x 7 F F F \equiv Q 15(1-\varepsilon)$ and $0 x 8000 \equiv$ Q15(-1).

## Mathematical notation

$z_{n}=x_{n} \times y_{n}$ with $n=0$ to ( $N-1$ ), while
$z_{n, \text { re }}=x_{n, r e} \cdot y_{n, r e}-x_{n, i m} \cdot y_{n, i m}$ and
$z_{n, i m}=x_{n, i m} \cdot y_{n, r e}+x_{n, r e} \cdot y_{n, i m}$

## 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, ${ }^{*} \mathrm{Z}$ is incremented with after each operation vector size N as a 16-bit unsigned value; defining the number of operations performed
Example for $\mathrm{Xi}=\mathrm{Yi}=\mathrm{Zi}=1, \mathrm{~N}=8$


## Uses

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


## Comments

- 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 $0 x 7 F F F \equiv Q 15(1-\varepsilon)$ and $0 x 8000 \equiv$ 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 $\mathrm{Xi}=\mathrm{Yi}=\mathrm{Zi}=2, \mathrm{~N}=16$


## 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 $\mathrm{Xi}=0, \mathrm{Yi}=0, \mathrm{Zi}=2$. (for gapless fill)
- Address mirrored additions and copy using negative $\mathrm{Xi}, \mathrm{Yi}$ or Zi
- Interleave/de-interleave and copy using other increment values


## Comments

- 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 $0 \times 7 F F F F F F F \equiv Q 31(1-\varepsilon)$ and $0 \times 80000000 \equiv$ Q31 $(-1)$.

## Mathematical notation

$\mathrm{z}_{\mathrm{n}}=\mathrm{x}_{\mathrm{n}}-\mathrm{y}_{\mathrm{n}}$ with $\mathrm{n}=0$ to ( $\mathrm{N}-1$ )

## 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 $X i$ as a 16-bit signed value, * $X$ is incremented with after each operation increment step of Yi as a 16 -bit signed value, ${ }^{*} \mathrm{Y}$ is incremented with after each operation increment step of Zi as a 16 -bit signed value, ${ }^{*} \mathrm{Z}$ is incremented with after each operation vector size N as a 16-bit unsigned value; defining the number of operations performed

Example for $\mathrm{Xi}=\mathrm{Yi}=\mathrm{Zi}=1 ; \mathrm{N}=8$


## 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 $\mathrm{Xi}=0, \mathrm{Yi}=0, \mathrm{Zi}=1$. (for gapless fill)
- Address mirrored additions and copy using negative $\mathrm{Xi}, \mathrm{Yi}$, or Zi
- Interleave/de-interleave and copy using other increment values


## Comments

- 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. Group 2 Parameters - Basic MAC Vector


### 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 $0 \times 7$ FFFFFFF $=$ Q31 $(1-\varepsilon)$ and $0 \times 80000000 \equiv$ Q31 $(-1)$.

## Mathematical notation

$$
Z=\sum_{n=0}^{N-1}\left(x_{n} * y_{n}\right)
$$

## 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 $\mathrm{N}=8$


## 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 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 $0 \times 7$ FFFFFFF $=$ Q31 $(1-\varepsilon)$ and $0 \times 80000000 \equiv$ Q31 $(-1)$.

## Mathematical notation

$$
\begin{array}{ll}
z_{0}=\sum_{n=0}^{N-2}\left(x_{n+1} * y_{n}\right) & \text { with } \mathrm{n}=0 \text { to }(\mathrm{N}-2) \\
z_{1}=\sum_{n=0}^{N-1}\left(x_{n} * y_{n}\right) & \text { with } \mathrm{n}=0 \text { to }(\mathrm{N}-1) \\
z_{2}=\sum_{n=0}^{N-2}\left(x_{n} * y_{n+1}\right) & \text { with } \mathrm{n}=0 \text { to }(\mathrm{N}-2)
\end{array}
$$

## 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 $\mathrm{Z1}\{\mathrm{~N}$ must be multiples of 2$\}$
Example for $\mathrm{N}=8$


### 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

$$
Z=\left(\sum_{n=0}^{N-1}\left(x_{n} \cdot y_{n}\right)\right) \cdot S F \quad \begin{aligned}
& \text { with } \mathrm{n}=0 \text { to (N-1) } \\
& \text { and } \mathrm{Z} \text { stored as IQ16. } 15 \text { (in an 32B/Q. } 31 \text { container) }
\end{aligned}
$$

with $\mathrm{n}=0$ to ( $\mathrm{N}-1$ ) 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 $\mathrm{N}=8$


## 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


### 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 $0 \times 7$ FFFFFFF $=$ Q31 $(1-\varepsilon)$ and $0 \times 80000000 \equiv$ Q31 ( -1 ).

## Mathematical notation

$$
\begin{aligned}
& z_{0}=\sum_{n=0}^{N-2}\left(x_{n} \cdot y_{n}\right) \quad \text { with } \mathrm{n}=0,2,6 \text { to }(\mathrm{N}-2) \\
& z_{1}=\sum_{n=1}^{N-1}\left(x_{n} \cdot y_{n}\right) \quad \text { with } \mathrm{n}=1,3,5 \text { to }(\mathrm{N}-1)
\end{aligned}
$$

## 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, ${ }^{*} \mathrm{Y}$ 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 $\mathrm{Xi}=\mathrm{Yi}=1 ; \mathrm{Zi}=3$; $\mathrm{N}=8$


## 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


## Comments

- 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 $0 \times 7$ FFFFFFF $=$ Q31 $(1-\varepsilon)$ and $0 \times 80000000 \equiv$ Q31 ( -1 ).

## Mathematical notation

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

## 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, ${ }^{*} \mathrm{X}$ is incremented with after each operation increment step of Yi as a 16-bit signed value, ${ }^{*} \mathrm{Y}$ is incremented with after each operation vector size N as a 16-bit unsigned value; defining the number of operations
Example for $\mathrm{Xi}=\mathrm{Y} \mathrm{i}=1$; $\mathrm{N}=8$


## 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


## Comments

- 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 $0 x 7 F F F F F F F \equiv Q 31(1-\varepsilon)$ and $0 \times 80000000 \equiv$ Q31 (-1).

## Mathematical notation

$$
\begin{aligned}
z & =\sum_{n=0}^{N-1}\left(x_{n} \cdot y_{n}\right) \text { with } \mathrm{n}=0 \text { to }(\mathrm{N}-1), \text { while } \\
z_{r e} & =\sum_{n=0}^{N-1}\left(x_{n, r e} \cdot y_{n, r e}-x_{n, i m} \cdot y_{n, i m}\right) \quad \text { and } \\
z_{i m} & =\sum_{n=0}^{N-1}\left(x_{n, i m} \cdot y_{n, r e}+x_{n, r e} \cdot y_{n, i m}\right)
\end{aligned}
$$

## 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, ${ }^{*} \mathrm{X}$ 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 $\mathrm{Xi}=\mathrm{Yi}=1 ; \mathrm{N}=8$


## 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


## Comments

- 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 $0 \times 7 F F F F F F F=$ Q31 $(1-\varepsilon)$ and $0 \times 80000000 \equiv$ Q31 $(-1)$.

## Mathematical notation

$$
\begin{aligned}
z & =\sum_{n=0}^{N-1}\left(x_{n} \cdot y_{n}^{*}\right) \quad \text { with } \mathrm{n}=0 \text { to }(\mathrm{N}-1), \text { while } \\
z_{r e} & =\sum_{n=0}^{N-1}\left(x_{n, r e} \cdot y_{n, r e}+x_{n, i m} \cdot y_{n, i m}\right) \quad \text { and } \\
z_{i m} & =\sum_{n=0}^{N-1}\left(x_{n, i m} \cdot y_{n, r e}-x_{n, r e} \cdot y_{n, i m}\right)
\end{aligned}
$$

## 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, ${ }^{*} \mathrm{X}$ is incremented with after each operation increment step of Yi as a 16-bit signed value, ${ }^{*} \mathrm{Y}$ is incremented with after each operation vector size N as a 16-bit unsigned value; defining the number of operations
Example for $\mathrm{Xi}=\mathrm{Yi}=1$; $\mathrm{N}=8$


## 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


## Comments

- 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.
Table 2-4. Group 4 Parameters: Basic Min/Max Vector Search on 16B Data

|  |  | $\xrightarrow{\text { 들 }}$ | - | $\begin{aligned} & \stackrel{N}{N} \\ & 0 \\ & 0 \\ & \hline 0 \\ & \hline 0 \\ & \hline \end{aligned}$ |  |  |
| :---: | :---: | :---: | :---: | :---: | :---: | :---: |
|  |  | $\begin{aligned} & V_{\text {So }}{ }^{*} \\ & =\& X \end{aligned}$ |  | $\begin{aligned} & \mathrm{Siz} \\ & =\mathrm{N} \end{aligned}$ | $\begin{aligned} & \mathrm{D}^{*} \\ & =\& \mathbf{Z} \end{aligned}$ | $\begin{gathered} z_{0}=\max _{n}\left\{x_{0}, \ldots, x_{n}\right\} ; \\ z_{1}=n \underset{\max }{\longrightarrow} z_{0} ; \end{gathered}$ |
| LEACMD__MAX | 0x104 | IQ16 |  | u16 | V,P |  |
| LEACMD__MAXUNSIGNED | 0x10C | u16 |  | u16 | V,P | $\begin{gathered} z_{0}=\min _{n}\left\{x_{0}, \ldots, x_{n}\right\} ; \\ z_{1}=n \underset{\min }{\longrightarrow} z_{0} ; \end{gathered}$ |
| LEACMD__MIN | 0x108 | IQ16 |  | u16 | V, P |  |
| LEACMD__MINUNSIGNED | 0x110 | u16 |  | u16 | V,P |  |

### 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

$$
\begin{aligned}
& z_{0}=\max _{n=0}^{N-1}\left(x_{n} \cdot y_{n}\right) \text { with } \mathrm{n}=0 \text { to }(\mathrm{N}-1), \text { and } \\
& z_{1}=n \xrightarrow[\max ]{ } z_{0}
\end{aligned}
$$

## 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

$$
\begin{aligned}
z_{0} & =\min _{n=0}^{N-1}\left(x_{n} \cdot y_{n}\right) \quad \text { with } \mathrm{n}=0 \text { to (N-1), and } \\
z_{1} & =n \xrightarrow[\min ]{ } z_{0}
\end{aligned}
$$

## 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

$$
\begin{aligned}
& z_{0}=\max _{n=0}^{N-1}\left(x_{n} \cdot y_{n}\right) \text { with } \mathrm{n}=0 \text { to }(\mathrm{N}-1), \text { and } \\
& z_{1}=n \xrightarrow[\max ]{ } z_{0}
\end{aligned}
$$

## 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

$$
\begin{aligned}
z_{0} & =\min _{n=0}^{N-1}\left(x_{n} \cdot y_{n}\right) \text { with } \mathrm{n}=0 \text { to }(\mathrm{N}-1), \text { and } \\
z_{1} & =n \xrightarrow[\min ]{ } z_{0}
\end{aligned}
$$

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

|  |  | $\begin{aligned} & \text { İ } \\ & \stackrel{0}{ㄹ} \end{aligned}$ |  | $\begin{aligned} & \stackrel{0}{N} \\ & \stackrel{N}{0} \\ & \hline \mathbf{O} \\ & \stackrel{O}{0} \\ & > \end{aligned}$ |  |  |  | $\begin{gathered} z_{0}=\max _{n}\left\{x_{0}, \ldots, x_{n}\right\} ; \\ z_{1}=n \underset{\max }{\longrightarrow} z_{0} ; \\ \mathrm{n}=0 \ldots \mathrm{~N}-1 \end{gathered}$ |
| :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: |
|  |  | $\begin{aligned} & \mathrm{V}_{\mathrm{SO}}{ }^{*} \\ & =\& \mathbf{X} \end{aligned}$ | $\begin{aligned} & \mathrm{J} \\ & 0 \\ & 0 \end{aligned}$ | $\begin{aligned} & \mathrm{Siz} \\ & =\mathrm{N} \end{aligned}$ | $\begin{array}{\|l\|l} \mathrm{D}^{\star} \\ =\& \mathbf{Z} \\ \hline \end{array}$ | $\begin{aligned} & V_{\mathrm{SO}+=. .} \\ & =\mathrm{X}_{\mathrm{i}} \end{aligned}$ |  |  |
| LEACMD__MAXLONGMATRIX | $0 \times \mathrm{D} 8$ | IQ32 |  | u16 | V,P | s16 | rsvd. |  |
| LEACMD__MAXUNSIGNEDLONGMATRIX | 0x114 | u32 |  | u16 | V,P | s16 | rsvd. | $z_{0}=\min _{n}\left\{x_{0}, \ldots, x_{n}\right\}$; |
| LEACMD__MINLONGMATRIX | $0 \times \mathrm{DC}$ | IQ32 |  | u16 | V,P | s16 | rsvd. | $z_{1}=n \underset{\min }{ } z_{0}$; |
| LEACMD__MINUNSIGNEDLONGMATRIX | 0x118 | u32 |  | u16 | V, P | s16 | rsvd. | $\mathrm{n}=0 . . . \mathrm{N}-1$ |

### 2.5.1 LEACMD__MAXLONGMATRIX (rMAXMI)

This command returns the value and position of the first maximum of vector $\mathbf{X}$ consisting of 32-bit signed numbers.

## Mathematical notation

$$
\begin{aligned}
& z_{0}=\max _{n=0}^{N-1}\left(x_{n} \cdot y_{n}\right) \quad \text { with } \mathrm{n}=0 \text { to }(\mathrm{N}-1), \text { and } \\
& z_{1}=n \xrightarrow[\max ]{ } z_{0}
\end{aligned}
$$

## 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 $X i$ 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 $\mathrm{Xi}=1$; $\mathrm{N}=8$


## Uses

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


## Comments

- 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 $\mathbf{X}$ consisting of 32-bit signed numbers.

## Mathematical notation

$$
\begin{aligned}
z_{0} & =\min _{n=0}^{N-1}\left(x_{n} \cdot y_{n}\right) \text { with } \mathrm{n}=0 \text { to }(\mathrm{N}-1), \text { and } \\
z_{1} & =n \xrightarrow[\min ]{ } z_{0}
\end{aligned}
$$

## 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, ${ }^{*} \mathrm{X}$ is incremented after each compare vector size $N$ as a 16-bit unsigned value; defining the length of vector
Example for $\mathrm{Xi}=1, \mathrm{~N}=8$


## Uses

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


## Comments

- 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 (rMAXMuI)

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

$$
\begin{aligned}
z_{0} & =\max _{n=0}^{N-1}\left(x_{n} \cdot y_{n}\right) \text { with } \mathrm{n}=0 \text { to }(\mathrm{N}-1), \text { and } \\
z_{1} & =n \xrightarrow[\max ]{ } z_{0}
\end{aligned}
$$

## 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 $\mathrm{Xi}=1, \mathrm{~N}=8$


## Uses

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


## Comments

- 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 (rMINMuI)

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

## Mathematical notation

$$
\begin{aligned}
& z_{0}=\min _{n=0}^{N-1}\left(x_{n} \cdot y_{n}\right) \text { with } \mathrm{n}=0 \text { to (N-1), and } \\
& z_{1}=n \xrightarrow[\text { min }]{ } z_{0}
\end{aligned}
$$

## 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, ${ }^{*} \mathrm{X}$ is incremented after each compare vector size $N$ as a 16-bit unsigned value; defining the length of vector
Example for $\mathrm{Xi}=1$; $\mathrm{N}=8$


## Uses

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


## Comments

- 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

|  |  | $\begin{aligned} & \bar{I} \\ & \underline{I} \end{aligned}$ |  | $\begin{aligned} & \stackrel{0}{0} \\ & \stackrel{0}{0} \\ & \stackrel{0}{0} \\ & \stackrel{y}{*} \end{aligned}$ | $\begin{aligned} & \stackrel{\rightharpoonup}{\stackrel{\omega}{0}} \\ & \stackrel{\rightharpoonup}{0} \\ & \stackrel{0}{7} \\ & \stackrel{\rightharpoonup}{=} \end{aligned}$ | $\begin{aligned} & \stackrel{\rightharpoonup}{7} \\ & \text { ? } \end{aligned}$ | $\begin{aligned} & \text { N } \\ & \stackrel{0}{0} \\ & \stackrel{\rightharpoonup}{0} \\ & \text { 区 } \\ & \hline \end{aligned}$ |  |  |
| :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: |
|  |  | $\begin{aligned} & \mathrm{v}_{\mathrm{SO}}{ }^{*} \\ & =\& \mathrm{x} \end{aligned}$ | $\begin{aligned} & \stackrel{\rightharpoonup}{3} \\ & 0 \\ & \hline \end{aligned}$ | $\begin{aligned} & \mathrm{Siz} \\ & =\mathrm{N} \end{aligned}$ | $\begin{aligned} & \mathrm{V}_{\mathrm{So}_{0}+=} \\ & =\mathrm{X}_{\mathrm{i}} \end{aligned}$ | $\begin{array}{r} \mathrm{D}^{*} \\ =\& Z \end{array}$ |  | $\begin{gathered} z_{0}=\max _{n}\left\{x_{0}, \ldots, x_{n}\right\} ; \\ \quad z_{2}=n \underset{\max }{\longrightarrow} z_{0} ; \end{gathered}$ | $\begin{gathered} z_{1}=\max _{n}\left\{x_{1}, \ldots, x_{n}\right\} ; \\ \\ z_{3}=n \xrightarrow{m a x} z_{1} ; \end{gathered}$ |
| LEACMD__MAXMATRIX | 0x44 | IQ16 |  | 416 | s16 | $\mathrm{V}_{\mathrm{E}}, \mathrm{V}_{0}, \mathrm{P}_{\mathrm{E},}, \mathrm{P}_{0}$ | rsvd. | $\mathrm{n}=0,2,4 \ldots \mathrm{~N}-2$ | $\mathrm{n}=1,3,5 \ldots . . \mathrm{N}-1$ |
| LEACMD__MAXUNSIGNEDMATRIX | 0xE8 | u16 |  | 416 | s16 | $V_{E}, V_{0}, P_{E}, P_{0}$ | rsvd. | $z_{0}=\min _{n}\left\{x_{0}, \ldots, x_{n}\right\}$; | $z_{1}=\min _{n}\left\{x_{1}, \ldots, x_{n}\right\}$; |
| LEACMD__MINMATRIX | $0 \times 48$ | IQ16 |  | 416 | s16 | $V_{E}, V_{0}, P_{E}, P_{0}$ | rsvd. | $z_{2}=n \overrightarrow{\text { min }} z_{0}$; | $z_{3}=n \xrightarrow[\text { min }]{ } z_{1}$; |
| LEACMD__MINUNSIGNEDMATRIX | $0 \times E C$ | u16 |  | u16 | s16 | $V_{E}, V_{0}, P_{E}, P_{0}$ | rsvd. | $\mathrm{n}=0,2,4 \ldots \mathrm{~N}-2$ | $\mathrm{n}=1,3,5 \ldots \mathrm{~N}-1$ |

### 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

$$
\begin{aligned}
& z_{0}=\max _{n=0}^{N-2}\left(x_{n} \cdot y_{n}^{*}\right) \quad \text { with } \mathrm{n}=0,2,6 \text { to }(\mathrm{N}-2), \text { and } \\
& z_{2}=n \xrightarrow[\max ]{ } z_{0} \\
& z_{1}=\max _{n=1}^{N-1}\left(x_{n} \cdot y_{n}^{*}\right) \text { with } \mathrm{n}=1,3,5 \text { to }(\mathrm{N}-1), \text { and } \\
& z_{3}=n \xrightarrow[\max ]{ } z_{1}
\end{aligned}
$$

## 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 $\mathrm{Xi}=1$; $\mathrm{N}=16$


## Uses

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


## Comments

- 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

$$
\begin{aligned}
& z_{0}=\min _{n=0}^{N-2}\left(x_{n} \cdot y_{n}^{*}\right) \text { with } \mathrm{n}=0,2,6 \text { to }(\mathrm{N}-2), \text { and } \\
& z_{2}=n \xrightarrow[\min ]{ } z_{0} \\
& \left.z_{1}=\min _{n=1}^{N-1}\left(x_{n} \cdot y_{n}^{*}\right) \text { with } \mathrm{n}=1,3,5 \text { to ( } \mathrm{N}-1\right), \text { and } \\
& z_{3}=n \xrightarrow[\min ]{ } z_{1}
\end{aligned}
$$

## 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 $\mathrm{Xi}=1$; $\mathrm{N}=16$


## Uses

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


## Comments

- 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

$$
\begin{aligned}
& z_{0}=\max _{n=0}^{N-2}\left(x_{n} \cdot y_{n}^{*}\right) \quad \text { with } \mathrm{n}=0,2,6 \text { to }(\mathrm{N}-2), \text { and } \\
& z_{2}=n \xrightarrow[\max ]{ } z_{0} \\
& z_{1}=\max _{n=1}^{N-1}\left(x_{n} \cdot y_{n}^{*}\right) \text { with } \mathrm{n}=1,3,5 \text { to }(\mathrm{N}-1), \text { and } \\
& z_{3}=n \xrightarrow[\max ]{ } z_{1}
\end{aligned}
$$

## 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 $\mathrm{Xi}=1$; $\mathrm{N}=16$


## Uses

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


## Comments

- 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

$$
\begin{aligned}
& z_{0}=\min _{n=0}^{N-2}\left(x_{n} \cdot y_{n}^{*}\right) \quad \text { with } \mathrm{n}=0,2,6 \text { to }(\mathrm{N}-2), \text { and } \\
& z_{2}=n \xrightarrow[\min ]{ } z_{0} \\
& z_{1}=\min _{n=1}^{N-1}\left(x_{n} \cdot y_{n}^{*}\right) \quad \text { with } \mathrm{n}=1,3,5 \text { to }(\mathrm{N}-1), \text { and } \\
& z_{3}=n \xrightarrow[\min ]{ } z_{1}
\end{aligned}
$$

## 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, ${ }^{*} \mathrm{X}$ is incremented (pairs) vector size N as a 16 -bit unsigned value; defining the length of vector \{multiple of 2$\}$
Example for $\mathrm{Xi}=1, \mathrm{~N}=16$


## Uses

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


## Comments

- 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.
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

$$
z_{n}=\sum_{k=0}^{K-1}\left(h_{k} \cdot x_{K-k+n-1}\right) \quad \text { with } \mathrm{n}=0 \text { to }(\mathrm{N}-1) \text { and } \mathrm{k}=0 \text { to (K-1) }
$$

## 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 $\mathrm{K}=8, \mathrm{~N}=6$, mask $=0 \times F F F F$


## 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 $\mathrm{K}+\mathrm{N}$ 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 \}

TEXAS
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 $=0 x F F F F$


## 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: $\bmod ($ address, blocksize $)=$ zero. When the circular buffers are used then the input data provision is also done in a circular addressed manner.


### 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

$$
z_{n}=\sum_{k=0}^{K-1}\left(h_{k} \cdot x_{K-k+n-1}\right) \text { with } \mathrm{n}=0 \text { to }(\mathrm{N}-1) \text { and } \mathrm{k}=0 \text { to }(\mathrm{K}-1)
$$

## 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 $\mathrm{K}=4, \mathrm{~N}=3$, mask $=0 x F F F F$


## 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 $\mathrm{K}+\mathrm{N}-1$ 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

INSTRUMENTS
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 $\mathrm{K}=8, \mathrm{~N}=6$, mask $=0 x F F F F$


## 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)


| mask value | suitable for... values |
| :--- | :--- |
| $0 \times 07(00000111)$ | :eight $Q .31$ values |
| $0 \times 0 F(00001111)$ | $: 16$ Q.31 values |
| $0 \times 1 F(00011111)$ | $: 32$ Q.31 values |
| $0 \times 3 F(00111111)$ | $: 64$ Q.31 values |
| $0 \times 7 F(01111111)$ | $: 128$ Q.31 values |
| $0 \times F F(11111111)$ | $: 256$ Q.31 values |

### 2.7.3 LEACMD

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

$$
\begin{aligned}
z_{n} & =\sum_{k=0}^{K-1}\left(h_{k} \cdot x_{K-k+n-1}\right) \quad \text { with } \mathrm{n}=0 \text { to }(\mathrm{N}-1) \text { and } \mathrm{k}=0 \text { to }(\mathrm{K}-1) \text {, while } \\
z_{n, r e} & =\sum_{k=0}^{K-1}\left(h_{k, r e} \cdot x_{K-k+n-1, r e}-h_{k, i m} \cdot x_{K-k+n-1, i m}\right) \quad \text { and } \\
z_{n, i m} & =\sum_{k=0}^{K-1}\left(h_{k, i m} \cdot x_{K-k+n-1, r e}+h_{k, r e} \cdot x_{K-k+n-1, i m}\right)
\end{aligned}
$$

## 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 $\mathrm{K}=4, \mathrm{~N}=3$, mask $=0 x F F F F$


## 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 $\mathrm{K}+\mathrm{N}-1$ 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)

TEXAS INSTRUMENTS

## 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 $\mathrm{K}=8, \mathrm{~N}=6$, mask $=0 x F F F F$


## 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 complex Q. 15 values |
| 0x0f $(00001111)$ | $: 16$ complex Q. 15 values |
| $0 \times 1 \mathrm{~F}(00011111)$ | $: 32$ complex Q. 15 values |
| $0 \times 3 F(00111111)$ | $: 64$ complex Q. 15 values |
| $0 \times 7 \mathrm{~F}(01111111)$ | $: 128$ complex Q.15 values |
| $0 \times F F(11111111)$ | $: 256$ complex Q. 15 values |

### 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

$$
\begin{aligned}
z_{n} & =\sum_{k=0}^{K-1}\left(h_{k} \cdot x_{K-k+n-1}\right) \quad \text { with } \mathrm{n}=0 \text { to }(\mathrm{N}-1) \text { and } \mathrm{k}=0 \text { to }(\mathrm{K}-1), \text { while } \\
z_{n, r e} & =\sum_{k=0}^{K-1}\left(h_{k, r e} \cdot x_{K-k+n-1, r e}-h_{k, i m} \cdot x_{K-k+n-1, i m}\right) \quad \text { and } \\
z_{n, i m} & =\sum_{k=0}^{K-1}\left(h_{k, i m} \cdot x_{K-k+n-1, r e}+h_{k, r e} \cdot x_{K-k+n-1, i m}\right)
\end{aligned}
$$

## 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 $\mathrm{K}=3, \mathrm{~N}=2$, mask $=0 x F F F F$


## 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 $\mathrm{K}+\mathrm{N}-1$ 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)

TEXAS INSTRUMENTS

## 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 $\mathrm{K}=8, \mathrm{~N}=6$, mask $=0 x F F F F$


## 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.


### 2.7.5 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

$$
z_{n}=\left(\sum_{k=0}^{K-1}\left(h_{k} \cdot x_{K-k+n-1}\right)\right) \cdot S F \quad \begin{array}{ll}
\text { with } \mathrm{n}=0 \text { to }(\mathrm{N}-1) \text { and } \mathrm{k}=0 \text { to }(\mathrm{K}-1) \text { and } \\
\mathrm{Z}_{\mathrm{n}} \text { stored as } \mathrm{Q} 16.15 \text { (in 32B/Q. } 31 \text { container) }
\end{array}
$$

## 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 $\mathrm{K}=8, \mathrm{~N}=6$, mask $=0 \times F F F F$


## 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 $\mathrm{K}+\mathrm{N}$ 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.
Table 2-8. Group 8 Parameters: Taylor McLaurin Functions on Pointwise Vectors or Matrices

|  |  | $\begin{aligned} & \stackrel{I}{Z} \\ & \stackrel{0}{I} \end{aligned}$ | $\begin{aligned} & \stackrel{్}{0} \\ & 0 \\ & D_{0} \\ & 0 \\ & 00 \end{aligned}$ | $\begin{aligned} & \text { O} \\ & \text { N } \\ & \text { No } \\ & \text { O} \\ & \text { © } \end{aligned}$ |  | $\stackrel{\leftrightarrow}{\otimes}$ | $\begin{aligned} & \text { © } \\ & \hline 0 . \end{aligned}$ | $\begin{aligned} & \overline{0} \\ & \stackrel{0}{\pi} \\ & \stackrel{\pi}{0} \\ & \underline{0} \end{aligned}$ |  |  |
| :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: |
|  |  | $\begin{aligned} & \mathrm{V}_{\text {SO }}{ }^{\prime} \\ & =\& \mathbf{X} \end{aligned}$ | $\begin{aligned} & \text { डुं } \\ & 0 \text { O} \end{aligned}$ | $\begin{aligned} & \mathrm{Siz} \\ & =\mathrm{N} \end{aligned}$ | $\begin{aligned} & V_{D}^{*} \\ & =\& \mathbf{Z} \end{aligned}$ | $\begin{aligned} & \mathrm{V}_{\mathrm{C}}{ }^{*} \\ & +\& \mathbf{C} \\ & \hline \end{aligned}$ | $\begin{array}{\|l\|} \hline \text { Ord } \\ =\mathrm{M} \\ \hline \end{array}$ | SF |  | $z_{n}=S F \sum^{M-1}\left(C_{m} * x_{n}^{m}\right)$ |
| LEASCCMD__POLYNOMIAL | $0 \times 20$ | Q. 15 |  | u16 | Q. 15 | Q. 15 | u16 | IQ16.15 |  |  |
| LEASCCMD__POLYNOMIALLONG | $0 \times 30$ | Q. 31 |  | u16 | Q. 31 | Q. 31 | 416 | s16 | rsvd. | $m=0$ $\mathrm{n}=0 . . \mathrm{N}-1$ |

### 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

$$
\begin{aligned}
& \left.z_{n}=\left(\sum_{m=0}^{M-1}\left(C_{m} \cdot x_{n}^{m}\right)\right) \cdot S F \quad \text { with } \mathrm{n}=0 \text { to }(\mathrm{N}-1) \text { and } \mathrm{m}=0 \text { to ( } \mathrm{M}-1\right) \text {, likewise } \\
& z_{n}=\left(c_{0}+c_{1} \cdot x_{n}+c_{2} \cdot x_{n}^{2}+c_{3} \cdot x_{n}^{3}+c_{4} \cdot x_{n}^{4} \ldots\right) \cdot S F \quad \text { with } \mathrm{n}=0 \text { to }(\mathrm{N}-1)
\end{aligned}
$$

## 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 $\mathrm{M}=8, \mathrm{~N}=16$


## 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

$$
\begin{aligned}
& z_{n}=\left(\sum_{m=0}^{M-1}\left(C_{m} \cdot x_{n}^{m}\right)\right) \cdot S F \quad \text { with } \mathrm{n}=0 \text { to }(\mathrm{N}-1) \text { and } \mathrm{m}=0 \text { to }(\mathrm{M}-1) \text {, likewise } \\
& z_{n}=\left(c_{0}+c_{1} \cdot x_{n}+c_{2} \cdot x_{n}^{2}+c_{3} \cdot x_{n}^{3}+c_{4} \cdot x_{n}^{4} \ldots\right) \cdot S F \quad \text { with } \mathrm{n}=0 \text { to (N-1) }
\end{aligned}
$$

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

|  |  |  |  |  |  |  |  |
| :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: |
| LEACMD_FFTCOMPLEXAUTOSCALING | $0 \times C 0$ | CQ. 15 |  | 416 | 416 |  |  |
| LEACMD__FFTCOMPLEXFIXEDSCALING | $0 \times 10$ | CQ. 15 |  | 416 | 416 | $=F F T\left(x_{0}, \ldots, x_{N-1}\right)$ with... | SF=> REG |
| LEACMD_FFFTCOMPLEXLONG | 0x9C | cQ. 31 |  | 416 | 416 | bit reverse sorted input | on AUTOSCALING |

INSTRUMENTS
www.ti.com

### 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 $\mathrm{N} / 2$ 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 $\mathrm{lb}(\mathrm{N})$ 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 $\mathrm{N} / 2$ 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 $\mathrm{lb}(\mathrm{N})$ 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.

INSTRUMENTS
www.ti.com

### 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: $\boldsymbol{m o d}($ 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 $\mathrm{N} / 2$ 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 $\mathrm{lb}(\mathrm{N})$ 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 $\mathrm{N} / 2$ 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 $\mathrm{lb}(\mathrm{N})$ 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 $\mathrm{N} / 2$ 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 $\mathrm{lb}(\mathrm{N})$ 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 $\mathrm{N} / 2$ 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 $\mathrm{lb}(\mathrm{N})$ 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

|  |  |  |  |  |  |
| :---: | :---: | :---: | :---: | :---: | :---: |
| LEACMD__BITREVERSECOMPLEXEVEN 0x84 | cQ. 15 |  | u16 | rsvd. | $\begin{gathered} \boldsymbol{X}=\operatorname{bit}-\operatorname{reverse}\left(x_{0}, \ldots, x_{N-1}\right) \\ \mathrm{n}=0 \ldots \mathrm{~N}-1 \text { with } \end{gathered}$ |
| LEACMD__BITREVERSECOMPLEXODD $0 \times 88$ | cQ. 15 |  | u16 | rsvd. |  |
| LEACMD__BITREVERSECOMPLEXLONGEVEN OxAC | cQ. 31 |  | u16 | rsvd. |  |
| LEACMD__BITREVERSECOMPLEXLONGODD 0xB4 | CQ. 31 |  | u16 | rsvd. |  |

### 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 $\operatorname{sqrt}(\mathrm{N})$ 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



### 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: $\bmod ($ 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



cBREVEIf ShuffleSize=4


## 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


### 2.11 Group 11: Post Operation for Real FFTs

Table 2-11 lists the parameter structure for this group.
Table 2-11. Group 11 Parameters: Post Operation for Real FFTs

|  |  |  |  |  |  | $\stackrel{N}{N}$ <br> 0 <br> 0 <br> O <br> O |  |
| :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: |
|  |  |  |  |  | $\begin{aligned} & \mathrm{Siz} \\ & \mathrm{~N} \end{aligned}$ | $\mathrm{lb}(\mathrm{N})$ |
| LEACMD | FFT | (FFTSPLIT) | 0x28 |  | Q. 15 |  | u16 | u16 |
| LEACMD | FFTLONG | (FFTLONGSPLIT) | 0xA4 | Q. 31 |  | u16 | u16 |

### 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: $\boldsymbol{m o d}($ 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)
$\log \operatorname{size} \mathrm{lb}(\mathrm{N})$ 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.
Table 2-12. Group 12 Parameters: De-Interleave and Sort Functions

|  |  |  | $\stackrel{\Gamma}{\square}$ |  |  | 䓂 |  |  |
| :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: |
|  |  | $\begin{aligned} & V_{\mathrm{SO}^{*}} \\ & =\& \mathrm{X} \\ & \hline \end{aligned}$ | $\begin{aligned} & \text { डु } \\ & \text { B } \\ & \hline \end{aligned}$ | $\begin{aligned} & \text { Siz } \\ & =N \end{aligned}$ | =M | $\begin{aligned} & \mathrm{V}_{\mathrm{D}}{ }^{2} \\ & =\& \mathbf{Z} \end{aligned}$ |  |  |
| LEACMD__DEINTERLEAVEEVENEVEN | 0x58 | IQ16 |  | u16 | u16 | IQ16 | rsvd. |  |
| LEACMD__DEINTERLEAVEEVENODD | $0 \times 5 \mathrm{C}$ | IQ16 |  | 416 | u16 | IQ16 | rsvd. |  |
| LEACMD__DEINTERLEAVEODDEVEN | $0 \times 60$ | IQ16 |  | u16 | u16 | IQ16 | rsvd. | $z_{n}=x_{n * M}$ |
| LEACMD__DEINTERLEAVEODDODD | 0x64 | IQ16 |  | u16 | u16 | IQ16 | rsvd. | -1 |
| LEACMD__DEINTERLEAVELONG | 0x12C | IQ32 |  | u16 | u16 | IQ32 | rsvd. |  |

InSTRUMENTS
www.ti.com

### 2.12.1 LEACMD__DEINTERLEAVEEVENEVEN (rSPLITEEs), <br> LEACMD__DEINTERLEAVEEVENODD (rSPLITEOs), LEACMD__DEINTERLEAVEODDEVEN (rSPLITOEs) <br> 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

$\mathrm{Z}_{\mathrm{n}}=\mathrm{X}_{\mathrm{n}}{ }^{*} \mathrm{M}$ with $\mathrm{n}=0$ to ( $\mathrm{N}-1$ )
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

$N=6, M=2$
blue: copied values
grey: unused elements


$N=6, M=4$
blue: copied values grey: unused elements

Examples with LEACMD__DEINTERLEAVEEVENODD


$$
\mathrm{N}=6, \mathrm{M}=3
$$



$N=4, M=5$
blue: copied values grey: unused elements

## Examples with LEACMD__DEINTERLEAVEODDEVEN


blue: copied values grey: unused elements


Examples with LEACMD__DEINTERLEAVEODDODD


Examples with LEACMD__DEINTERLEAVELONG

blue: copied values



$$
\mathrm{N}=4, \mathrm{M}=7
$$

blue: copied values grey: unused elements


$$
N=2, M=9
$$

blue: copied values grey: unused elements

## 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


## Common de-interleave pattern



### 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


### 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 $\mathrm{N}=3$


TEXAS
INSTRUMENTS
www.ti.com

### 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 Stack Frame Structure

| Memory Address | Register Content Stored |
| :---: | :---: |
| LEASPTR + 100 | LEADSTSTAT |
| LEASPTR + 96 | LEACNF0 |
| LEASPTR + 92 | LEACNF1 |
| LEASPTR + 88 | LEACMDSTAT |
| LEASPTR + 84 | LEAS1STAT |
| LEASPTR + 80 | LEASOSTAT |
| 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


TEXAS
INSTRUMENTS

### 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


### 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

|  |  |  |  |  |
| :---: | :---: | :---: | :---: | :---: |
| * | 0x1300 | 0xE900 | 0x1D00 | 0xE100 |
| ${ }^{*} X_{3 r d} \cdots$ | 0xD800 | 0x2900 | 0xD500 | 0x2F00 |
|  | 0x3500 | 0xC500 | 0x3D00 | 0xBD00 |
|  | 0xB900 | 0x4900 | 0xB100 | 0x5300 |
|  | 0x5900 | 0x9F00 | 0x6500 | 0x9900 |
|  | 0x9500 | 0x6D00 | 0x8F00 | 0x7F00 |



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


### 2.14.2 LEACMD

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

$$
\begin{aligned}
\sim \text { PMDST } & =\left(\sum_{m=0}^{M-1}\left(C_{m} \cdot x_{n}^{m}\right)\right) \cdot S F \quad \text { with } \mathrm{n}=0 \text { to }(\mathrm{N}-1) \text { and } \mathrm{m}=0 \text { to }(\mathrm{M}-1) \\
\text { LEAPMDST } & =\left(c_{0}+c_{1} \cdot x_{n}+c_{2} \cdot x_{n}^{2}+c_{3} \cdot x_{n}^{3}+c_{4} \cdot x_{n}^{4} \ldots\right) \cdot S F \quad \text { with } \mathrm{n}=0 \text { to }(\mathrm{N}-1)
\end{aligned}
$$

## 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 $\mathrm{M}=4$

purple: indicating start index
blue: tributaries for first result

## 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

$$
H(z)=\frac{b 0+b 1 * z^{-1}+b 2 * z^{-2}}{1+a 1 * z^{-1}+a 2 * z^{-2}}
$$

## Implemented function

$$
H(z)=\frac{B 0+B 1 * z^{-1}+B 2 * z^{-2}}{1-A 1 * z^{-1}-A 2 * z^{-2}}
$$

## 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 $B Q$ 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 $0,1,4,5,8,9, \ldots$ ).

Example for $\mathrm{N}=6$, 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 $B 1$ is added twice. This also can be seen that the value for $A 1$ and $B 1$ is interpreted as Q1.14 format, Also true is that A 1 and B 1 holds half the value in Q .15 format.

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

INSTRUMENTS

### 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 $A 0, A 1, A 2, B 0, B 1$ and $B 2$ 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

$$
H(z)=a 0 \quad \frac{b 0+b 1 * z^{-1}+b 2 * z^{-2}}{1+a 1 * z^{-1}+a 2 * z^{-2}}
$$

## Implemented function

$$
H(z)=A 0 \frac{B 0+B 1 * z^{-1}+B 2 * z^{-2}}{1-A 1 * Z^{-1}-A 2 * Z^{-2}}
$$

## 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 ( $\mathrm{N}=16$ will calculate 32 samples of Q.15)

Example for $\mathrm{N}=6$


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 $\mathrm{A} 0, \mathrm{~A} 1, \mathrm{~A} 2, \mathrm{~B} 0, \mathrm{~B} 1$ and B 2 are derived from those given in many text books ( $\mathrm{a} 0, \mathrm{a} 1, \mathrm{a} 2, \mathrm{~b} 0, \mathrm{~b} 1, \mathrm{~b} 2$ ). A simple scaling step allows to simplify while maintaining high accuracy. It allows A 0 to be smaller than one and A2 to reach -1 without epsilon offset. A1 and B1 range from -2 to $2-2 e p s$.

## Mathematical notation of addressed function

$$
H(z)=a 0 \quad \frac{b 0+b 1 * z^{-1}+b 2 * z^{-2}}{1+a 1 * z^{-1}+a 2 * z^{-2}}
$$

## Implemented function

$$
H(z)=B i a s+A 0 \frac{B 0+B 1 * Z^{-1}+B 2 * Z^{-2}}{1-A 1 * Z^{-1}-A 2 * Z^{-2}}
$$

## 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 $B Q$ filter element vector size $N$ as a 16-bit unsigned value, defining number of value pairs processed ( $\mathrm{N}=16$ will calculate 32 samples of Q.15)

Example for $\mathrm{N}=6$


Note: The minima and maxima of the even indexes samples are maintained in MAXO/MINO.
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

$$
\begin{aligned}
& \text { for } n=0 \ldots K-1 \\
& \qquad x_{N-n}=x_{N-n} * c_{n} \\
& \text { and } x_{n}=x_{n} * c_{n}
\end{aligned}
$$

## 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 $\mathrm{N}=12, \mathrm{~K}=4$


InSTRUMENTS
www.ti.com
LEA Hardware

## 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.

Table 3-1. LEA Common Functions Polynomial Coefficients

| Name | Description | Address \{LEA\} | Width | Size N |
| :---: | :---: | :---: | :---: | :---: |
| LEA_CONST_LONG_SQRT | coefficients for SQRT(X-1) | 0x8000 | 32B | 33 |
| LEA_CONST_LONG_EXP | coefficients for EXP (X) | $0 \times 8021$ | 32B | 13 |
| LEA_CONST_LONG_2COS | coefficients for $\operatorname{COS}\left(\mathrm{X}^{\wedge} 2\right)$ | 0x802E | 32B | 7 |
| 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 | $0 \times 8088$ | 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( $\left.\mathrm{X}^{\wedge} 2\right)$ | 0x80FE | 32B | 10 |
| LEA_CONST_LONG_LOG | coefficient for LOG(X) | 0x8108 | 32B | 13 |
| LEA_CONST_LONG_4EXP | coefficient for EXP $(X)$ auxiliary | $0 \times 8115$ | 32B | 10 |
| LEA_CONST_LONG_2SIN | coefficient for $\operatorname{SIN}\left(\mathrm{X}^{\wedge} 2\right)$ | 0x811F | 32B | 4 |

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 <br> positive amplitude margin 16B, + 0.5 | 0x4000 | $0 x A 800$ to 0xABFF | 16B (dual) | 2048 |
| LEA_CONST_7 neutral element, +1 | 0x7FFF | 0xAC00 to 0xAFFF | 16B (dual) | 2048 |
| LEA_CONST_8 <br> substitutions, complementary element, conversion of ADC/DAC formats, $-1, \ldots$ | 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 | 0xFFFFE | $0 x B 800$ 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 | $0 \times B C 00$ to 0xBFFF | 16B (dual) | 2048 |
| LEA_CONST_LONG_0 neutral element, dummy source | 0x0 | $0 x C 000$ to 0xC7FF | 32B | 2048 |
| LEA_CONST_LONG_1 positive epsilon | 0x1 | $0 x C 800$ to 0xCFFF | 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 0xFFFFF | 32B | 2048 |

### 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.


Figure 3-1. LEA Timer Block Diagram

### 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.


Registers mainly for Programming Purposes
Registers mainly for Debug and Diagnostics Purposes

Figure 3-2. Overview of LEA hardware Registers

## 4 Revision History

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

| DATE | REVISION |  |
| :--- | :--- | :--- |
| 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 Tl 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.

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

