RADARLIB IMPLEMENTATION ON EVE CO-PROCESSOR

TEXAS INSTRUMENTS INC.

Table of Contents

RADARLIB IMPLEMENTATION ON EVE CO-PROCESSOR 1
Table of Contents 2
1. Interference Zero Out 3
Description 3
Location 3
Usage 3
2. DC offset and Windowing 4
Description 4
Location 4
Usage 4
Performance Considerations 8
3. FFT 9
Description 9
Location 9
Usage 9
4. Beam Forming 16
Description 16
Location 16
Usage 16
5. Peak Detection 23
Description 23
Location 23
Usage 23

NOTE: It should be noted that some APIs in this document contains performance and code size information – These are only theoretical estimates, Please refer to the data sheet for the actual performance and the code size.

1. Interference Zero Out

Description

The routine accepts a 16 bit complex input data and returns a 16 bit complex output data such that each location whose absolute value is greater than user specified threshold is set to zero.. Following kernel is used for this feature :

Location

<EVE_SW_ROOT>/kernels/radarlib/vcop_dcoffset_windowing

Usage

```
void vcop_interference_zero_out_kernel
(
  __vptr_int16   inputData,
  __vptr_int16   outputData,
  unsigned short interferenceThreshold,
  unsigned short numPoints,
  unsigned short numOfLines,
  unsigned short outputPitch
)
```

@inputs This kernel takes following Inputs

inputData :

Input buffer containing data 16 bit signed data with real and imaginary part interleaved.

Size of this buffer should be numPoints * numOfLines * sizeof(int16_t) * 2

interferenceThreshold :

Interference threshold to be used

numPoints :

Number of points whose dc offset calculation is required.

numOfLines :

Number of lines to work with in single kernel

outputPitch :

Number of bytes to jump from one line to next line in the output buffer

@scratch This kernel needs following scratch buffers

@outputs This kernel produce following outputs
outputData
Pointer to the output buffer containing the output of this kernel which is stored with real and imaginary part interleaved .
Size of this buffer should be is same as input buffer size which is numPoints * numOfLines * sizeof(int16_t) * 2

@remarks Following is the buffer placement assumed for optimal performance of this kernel
inputData and outputData should be placed in two different buffer for optimal performance

2. DC offset and Windowing

Description

These set of kernels are involved in calculating DC offset and windowing operation. DC offset is nothing but the average value of a line. Input to DC offset kernel is a 16-bit signed complex data and output is set of average value for each line of the input. Windowing kernel multiplies a 16bit signed complex number with a 16bit signed real number and store the output as 16-bit signed complex number after applying rounding. Another version of windowing kernel writes the final output in transposed format.

Location

<EVE_SW_ROOT>/kernels/radarlib/vcop_dcoffset_windowing

Usage

This routine is C-callable and can be called as :

```
void vcop_dcoffset_kernel
(
  __vptr_int16   inputData,
  __vptr_int32   scratchBuf,
  __vptr_uint16  pScatterIndex,
  __vptr_int16   dcOffsetBuf,
  unsigned short transposeStride,
  unsigned short numPoints,
  unsigned short numOfLines,
  unsigned short shift
)
```

@inputs This kernel takes following Inputs
input Data:
Input buffer containing data 16 bit signed data with real and imaginary part Interleaved.
Size of this buffer should be numPoints * numOfLines * sizeof(int16_t) * 2
numPoints:
Number of points whose dc offset calculation is required.

numOfLines:
Number of lines to work with in single kernel
shift:
Amount of shift that needs to be applied in final output
pScatterIndex:
Pointer to the 8 indexes to be used for pScatter. Each entry of this should be such that all 8 lanes of a vector goes to different memory bank. Size of this buffer should be VCOP_SIMD_WIDTH * sizeof(int16_t)

@scratch This kernel needs following scratch buffers
scratchBuf :
Pointer to the scratch buffer .
Size of this buffer should be transposeStride * sizeof(int32Pt) * VCOP_SIMD_WIDTH

@outputs This kernel produce following outputs
dcOffsetBuf
Pointer to the output buffer containing the output of this kernel which is stored with real and imaginary part interleaved . Total number of elements would be equal to number of lines. Size of this buffer should be ((numOfLines) * size(int16_t) * 2)

@remarks Following is the buffer placement assumed for optimal performance of this kernel
Compute bound case so buffer placement could be anywhere

@constraints Following constraints
None

@return NONE

```
void vcop_windowing_kernel
(
  __vptr_int16        inputData,
  __vptr_int16        winCoefficients,
  __vptr_int16        dcOffsetBuf,
  __vptr_int16        outputData,
  unsigned short      numPoints,
  unsigned short      numOfLines,
  unsigned short      scaleFactor,
  unsigned short      saturationLimit,
  unsigned short      outputPitch
)
```

@desc This function applyies a scaling factor to each entery of input data for both real and imagignary parts

@inputs This kernel takes following Inputs

inputData :
Input buffer containing data 16 bit signed data with real and imaginary part interleaved.
Size of this buffer should be numPoints * numOfLines * sizeof(int16_t) * 2
winCoefficients :
Buffer which holds the scalar window coefficients to apply
Size of this buffer should be numPoints * sizeof(int16_t)
dcOffsetBuf :
Buffer to the DC offsets for each line
numPoints :
Number of points whose windowing is required.
numOfLines :
Number of lines to work with in single kernel
saturationLimit :
Limit the output to certain range. Output will be limited with -saturationLimit to (saturationLimit - 1)
outputPitch :
Number of bytes to jump from one line to next line in the output buffer

@scratch This kernel needs following scratch buffers

@outputs This kernel produce following outputs
outputData
Pointer to the output buffer containing the output of this kernel
Size of this buffer should be

@remarks Following is the buffer placement assumed for optimal performance of this kernel
inputdata, winCoefficients , outputData one of these three buffer should be iin different memory

@constraints Following constraints
None

@return NONE

```
void vcop_windowing_with_transpose_kernel
(
  __vptr_int16        inputData,
  __vptr_int16        winCoefficients,
  __vptr_int16        outputData,
  __vptr_uint16       pScatterIndex,
  unsigned short      transposePitch,
  unsigned short      numPoints,
  unsigned short      numOfLines,
  unsigned short      scaleFactor ,
  unsigned short      saturationLimit)
```

@desc This function applyies a scaling factor to each entery of input data for both real and imagignary parts and
stores the output in tranpose order

@inputs This kernel takes following Inputs
inputData :
Input buffer containing data 16 bit signed data with real and imaginary part interleaved.
Size of this buffer should be numPoints * numOfLines * sizeof(int16_t) * 2
winCoefficients :
Buffer which holds the scalar window coefficients to apply
Size of this buffer should be numPoints * sizeof(int16_t)
numPoints :
Number of points
numOfLines :
Number of lines to work with in single kernel
pScatterIndex :
Pointer to the index to be used for pScatter. This should be such that all 8 lanes of a vector goes to different memory bank.
transposePitch :
Pitch in bytes for the pitch to be used for transpose
saturationLimit :
Limit the output to certain range. Output will be limited with -saturationLimit to (saturationLimit - 1)

@scratch This kernel needs following scratch buffers
None
@outputs This kernel produce following outputs
outputData
Pointer to the output buffer containing the output of this kernel
Size of this buffer should be (transposePitch * numPoints)

@remarks Following is the buffer placement assumed for optimal performance of this kernel
Compute bound case so buffer placement could be anywhere

@constraints Following constraints
None

@return NONE

```
void vcop_sign_extension_kernel
(
  __vptr_uint16  inputData,
  __vptr_int16   outputData,
  unsigned short numPoints,
  unsigned short numOfLines,
  unsigned short outputPitch,
  unsigned short inBits
)

@kernel    vcop_sign_extension_kernel

  @desc     This function extends the sign of input data from n bits to 16 bits. where n < 16

  @inputs   This kernel takes following Inputs
            inputData :
                  Input buffer containing data 16 bit signed data with real and imaginary part
                  interleaved.
                  Size of this buffer should be numPoints * numOfLines * sizeof(int16_t) * 2
            numPoints :
                  Number of points
            numOfLines :
                  Number of lines to work with in single kernel
           outputPitch :
                  Number of bytes to jump from one line to next line in output buffer
           inBits :
                  Number of bits valid bits in input data which is stored in 16b it container

  @scratch   This kernel needs  following scratch buffers

  @outputs   This kernel produce following outputs
            outputData
                  Pointer to the output buffer containing the output of this kernel
                  Size of this buffer should be numPoints * numOfLines * sizeof(int16_t) * 2

  @remarks  Following is the buffer placement assumed for optimal performance of this kernel
                  Compute bound case so buffer placement could be anywhere

  @constraints Following constraints
                  None

  @return    NONE
```

Performance Considerations

- Please refer the kernel test bench code for buffer placement

3. FFT

Description

These set of kernels are optimized implementation of various stages of various N- Point FFT. Currently supported values of N are (64, 128, 256, 512, 1024).

Location

<EVE_SW_ROOT>/kernels/radarlib/vcop_fft_npt_16ix16o

Usage

This routine is C-callable and can be called as:

```
void vcop_fft_1024_16i_16o(
    int16_t      * pInputDataWBuf,
    int16_t      * pScratchIBufL,
    int16_t      * pScratchIBufH,
    int16_t      *twiddleFactorBuf,
    uint16_t     *pScatterOffset,
    uint16_t     pitch,
    uint16_t     scaleFactorArray[],
    uint16_t     numOfLines,
    uint8_t      enableInPlaceCompute);
```

@desc This is a wrapper function of init and vloop for all FFT stages for 1024 point. This will initialize all the param blocks;

@inputs This kernel takes following Inputs

pInputDataWBuf:

Input buffer containing data 16 bit signed data with real and imaginary part interleaved. This buffer is expected to be in WBUF

Refer testbench to know the amount of memory required for this

pTwiddleFactor :

Buffer which holds twidlde factor for this kernel implementaion. The order in which these are generated can be seen from vcop_fft_npt_16ix16o_gen_twiddleFactor.c file

Size of this buffer should be getSizeTwiddleFactor_1024 ()

pScatterOffset :

Buffer which stores 8 indexes to be used for doing transpose. Please refer the testbench to check how this is calculated :

scaleFactorArray :

Scale factor to be used for reach stage

numOfLines :

Number of lines for which this processing needs to be done

pitch :

Offset in terms of number of bytes to move from one line to the next line. Pitch should be word aligned

enableInPlaceCompute : If enabled the kernel will overwrite the input data itself otherwise output would be written topScratchIBufH

@scratch This kernel needs following scratch buffers

pScratchIBufL :

Scratch buffer in Image buffer L which store intermediate FFT stage result. Refer testbench to know the amount of memory required for this.

pScratchIBufH:

Scratch buffer in Image buffer H which store intermediate FFT stage result. Refer testbench to know the amount of memory required for this result

@outputs This kernel produce following outputs

Output of the kernel is depends on enableInPlaceCompute

@remarks Following is the buffer placement assumed for optimal performance of this kernel

pInput :IBUFLA/WBUF

pOutput :IBUFHA

@constraints Following constraints

None

@return NONE

```
void vcop_fft_512_16i_16o(
   int16_t      * pInputDataWBuf,
   int16_t      * pScratchIBufL,
   int16_t      * pScratchIBufH,
   int16_t      *twiddleFactorBuf,
   uint16_t     *pScatterOffset,
   uint16_t     scaleFactorArray[],
   uint16_t     numOfLines,
   uint8_t       enableInPlaceCompute)
```

@desc This is a wrapper function of init and vloop for all FFT stages for 512 point. This will initialize all the param blocks;

@inputs This kernel takes following Inputs

pInputDataWBuf:

Input buffer containing data 16 bit signed data with real and imaginary part interleaved. This buffer is expected to be in WBUF

Refer testbench to know the amount of memory required for this

pTwiddleFactor :

Buffer which holds twidlde factor for this kernel implementaion. The order

in which these are generated can be seen from
vcop_fft_npt_16ix16o_gen_twiddleFactor.c file
Size of this buffer should be getSizeTwiddleFactor_512()
pScatterOffset :
Buffer which stores 8 indexes to be used for doing transpose. Please refer the testbenchto check how this is calculated

scaleFactorArray :
Scale factor to be used for reach stage
numOfLines :
Number of lines for which this processing needs to be done
enableInPlaceCompute : If enabled the kernel will overwrite the input data itself otherwise output would be written top ScratchIBufH

@scratch This kernel needs following scratch buffers
pScratchIBufL :
Scratch buffer in Image buffer L which store intermediate FFT stage result.
Refer testbench to know the amount of memory required for this.
pScratchIBufH:
Scratch buffer in Image buffer H which store intermediate FFT stage result.
Refer testbench to know the amount of memory required for this result.

@outputs This kernel produce following outputs
inputData :
This kernel overwrites the input data itself for output
Size of this buffer should be numPoints * numOfLines * sizeof(int16_t) * 2

@remarks Following is the buffer placement assumed for optimal performance of this kernel
pInput :IBUFLA/WBUF
pOutput :IBUFHA

@constraints Following constraints
None

@return NONE

```
void vcop_fft_128_16i_16o(
  int16_t     * pInputDataWBuf,
  int16_t     * pScratchIBufL,
  int16_t     * pScratchIBufH,
  int16_t     *twiddleFactorBuf,
  uint16_t    *pScatterOffset,
  uint16_t    pitch,
  uint16_t    scaleFactorArray[],
  uint16_t    numOfLines,
  uint8_t     enableInPlaceCompute);
```

@desc This is a wrapper function of init and vloop for all FFT stages for 128 point. This will initialize all the param blocks;

@inputs This kernel takes following Inputs

pInputDataWBuf:

Input buffer containing data 16 bit signed data with real and imaginary part interleaved. This buffer is expected to be in WBUF
Refer testbench to know the amount of memory required for this

pTwiddleFactor :

Buffer which holds twidlde factor for this kernel implementaion. The order in which these are generated can be seen from vcop_fft_npt_16ix16o_cn.c file
Size of this buffer should be getSizeTwiddleFactor_128()

pScatterOffset :

Buffer which stores 8 indexes to be used for doing transpose. Please refer the testbench to check how this is calculated :

scaleFactorArray :

Scale factor to be used for reach stage

numOfLines :

Number of lines for which this processing needs to be done

pitch :

Offset in terms of number of bytes to move from one line to the next line. Pitch should be word aligned

enableInPlaceCompute : If enabled the kernel will overwrite the input data itself otherwise output would be written topScratchIBufH

@scratch This kernel needs following scratch buffers

pScratchIBufL :

Scratch buffer in Image buffer L which store intermediate FFT stage result.
Refer testbench to know the amount of memory required for this.

pScratchIBufH:

Scratch buffer in Image buffer H which store intermediate FFT stage result.
Refer testbench to know the amount of memory required for this result.

@outputs This kernel produce following outputs

inputData :

This kernel overwrites the input data itself for output
Size of this buffer should be numPoints * numOfLines * sizeof(int16_t) * 2

@remarks Following is the buffer placement assumed for optimal performance of this kernel

pInput :IBUFLA/WBUF
pOutput :IBUFHA

@constraints Following constraints

None

@return NONE

```
void vcop_fft_256_16i_16o(
    int16_t      * pInputDataWBuf,
    int16_t      * pScratchIBufL,
    int16_t      * pScratchIBufH,
    int16_t      *twiddleFactorBuf,
    uint16_t     *pScatterOffset,
    uint16_t     pitch,
    uint16_t     scaleFactorArray[],
    uint16_t     numOfLines,
    uint8_t      enableInPlaceCompute);
```

@func vcop_fft_256_16i_16o

@desc This is a wrapper function of init and vloop for all FFT stages for 256 point. This will initialize all the param blocks;

@inputs This kernel takes following Inputs

pInputDataWBuf:

Input buffer containing data 16 bit signed data with real and imaginary part interleaved. This buffer is expected to be in WBUF

Refer testbench to know the amount of memory required for this

pTwiddleFactor :

Buffer which holds twidlde factor for this kernel implementaion. The order in which these are generated can be seen from vcop_fft_npt_16ix16o_cn.c file

Size of this buffer should be getSizeTwiddleFactor_256()

pScatterOffset :

Buffer which stores 8 indexes to be used for doing transpose. Please refer the testbench

to check how this is calculated :

scaleFactorArray :

Scale factor to be used for reach stage

numOfLines :

Number of lines for which this processing needs to be done

pitch :

Offset in terms of number of bytes to move from one line to the next line. Pitch should be word aligned

enableInPlaceCompute : If enabled the kernel will overwrite the input data itself otherwise output would be written to pScratchIBuf

@scratch This kernel needs following scratch buffers

pScratchIBufL :

Scratch buffer in Image buffer L which store intermediate FFT stage result.

Refer testbench to know the amount of memory required for this.

pScratchIBufH:

Scratch buffer in Image buffer H which store intermediate FFT stage result. Refer testbench to know the amount of memory required for this result.

@outputs This kernel produce following outputs

inputData :

This kernel overwrites the input data itself for output

Size of this buffer should be numPoints * numOfLines * sizeof(int16_t) * 2

@remarks Following is the buffer placement assumed for optimal performance of this kernel

pInput :IBUFLA/WBUF

pOutput :IBUFHA

@constraints Following constraints

None

@return NONE

```
void vcop_fft_64_16i_16o (
   int16_t      *pInputDataWBuf,
   int16_t      *pScratchIBufL,
   int16_t      *pScratchIBufH,
   int16_t      twiddleFactorBuf[],
   uint16_t     *pScatterOffset,
   uint16_t     pitch,
   uint16_t     scaleFactorArray[],
   uint16_t     numOfLines,
   uint8_t      enableInPlaceCompute);
```

@func vcop_fft_64_16i_16o

@desc This is a wrapper function of init and vloop for all FFT stages for 64 point. This will initialize all the param blocks;

@inputs This kernel takes following Inputs

pInputDataWBuf :

Input buffer containing data 16 bit signed data with real and imaginary part interleaved.

Size of this buffer should be numPoints * numOfLines * sizeof(int16_t) * 2

pTwiddleFactor :

Buffer which holds twidlde factor for this kernel implementaion. The order in which these are generated can be seen from vcop_fft_npt_16ix16o_cn.c file

Size of this buffer should be getSizeTwiddleFactor_64()

pScatterOffset :

Buffer which stores 8 indexes to be used for doing transpose. Please refer the testbench to check how this is calculated :

scaleFactorArray :
Scale factor to be used for reach stage
numOfLines :
Number of lines for which this processing needs to be done
pitch :
Offset in terms of number of bytes to move from one line to the next line. Pitch should be word aligned
enableInPlaceCompute : If enabled the kernel will overwrite the input data itself otherwise output would be written to pScratchIBuf

@scratch This kernel needs following scratch buffers
pScratchIBufL :
Scratch buffer in Image buffer L which store intermediate FFT stage result.
Refer testbench to know the amount of memory required for this.
pScratchIBufH:
Scratch buffer in Image buffer H which store intermediate FFT stage result.
Refer testbench to know the amount of memory required for this

@outputs This kernel produce following outputs
Output of the kernel is depends on enableInPlaceCompute

@remarks Following is the buffer placement assumed for optimal performance of this kernel
pInput :IBUFLA/WBUF
pOutput :IBUFHA

@constraints Following constraints
None

@return NONE

4. Beam Forming

Description

These set of kernels are optimized implementation of various kernels required for beam forming.

Location

<EVE_SW_ROOT>/kernels/radarlib/vcop_beam_forming

Usage

This routine is C-callable and can be called as:

```
vcop_beam_forming_copy_steering_matrix_kernel
(
  __vptr_uint32   inputData,
  __vptr_uint32   outputData,
  unsigned short  numAngles,
  unsigned short  numAntennas
);
```

@desc This kernel copies the steering matrix from one buffer to other

@inputs This kernel takes following Inputs

inputData :

Input buffer containing data 16 bit signed data with real and imaginary part interleaved. Input buffer is of dimension numAngles x numAntennas.

Size of this buffer should be numAngles * numAntennas * sizeof(int16_t) * 2

numAngles :

Number of angles to be determined. This should come from the resolution user wants

numAntennas :

Total number of antennnas

@scratch This kernel needs following scratch buffers

@outputs This kernel produce following outputs

outputData

Pointer to the output buffer containing the output of this kernel which is stored with real and imaginary part interleaved .

Size of this buffer should be ((numAngles * numAngles) * size(int16_t) * 2)

@remarks Following is the buffer placement assumed for optimal performance of this kernel

inputData, outputData should be in different memory

@constraints Following constraints

@return NONE

void vcop_beam_forming_transpose_antenna_data_kernel
(
__vptr_uint32 inputData,
__vptr_uint32 outputData,
__vptr_uint16 pScatterIndex,
unsigned short outputPitch,
unsigned short numDetections,
unsigned short numAntennas
)

@desc This kernel tranpose the antenna data

@inputs This kernel takes following Inputs
inputData :
Input buffer containing data 16 bit signed data with real and imaginary part interleaved. Input buffer is of dimension numAntennas x numDetections.
Size of this buffer should be numDetections * numAntennas * sizeof(int16_t) * 2
numDetections :
Number of angles to be determined. This should come from the resolution user wants
numAntennas :
Total number of antennnas
outputPitch :
Pitch at which transposed output will be stored

@scratch This kernel needs following scratch buffers

@outputs This kernel produce following outputs
outputData
Pointer to the output buffer containing the output of this kernel which is stored with real and imaginary part interleaved .
Size of this buffer should be ((numDetections*numAntennas) * size(int16_t)* 2)

@remarks Following is the buffer placement assumed for optimal performance of this kernel
inputData, outputData should be in different memory

@constraints Following constraints

@return NONE

```
void vcop_beam_forming_kernel
(
  __vptr_int16    inputData,
  __vptr_int16    outputData,
  __vptr_int16    steeringMatrix,
  unsigned short  pitch,
  unsigned short  numDetections,
  unsigned short  numAntennas,
  unsigned short  numAngles,
  unsigned short  scale
);
```

@desc This kernel does the beam forming which is esstentially a matrix multiplication of input data (numDetections x numAntennas) with steering matrix (numAntennas x numAngles)

@inputs This kernel takes following Inputs

inputData :

Input buffer containing data 16 bit signed data with real and imaginary part interleaved. Input buffer is of dimension numDetections x numAntennas, here detections are in horizontal direction and anetnna data is in vertical direction. Size of this buffer should be numDetections*numAntennas * sizeof(int16_t) * 2

steeringMatrix :

Steering matrix for all the antenna. Dimension of this buffer is numAntennas x numAngles, here antenna data is in horizontal direction and angle data is in vertical direction. Size of this buffer should be numAntennas*numAngles * sizeof(int16_t) * 2

pitch :

Pitch in bytes to access next line inputData

numDetections :

Number of detections whose angle needs to be determined

numAntennas :

Total number of antennnas

numAngles :

Number of angles to be determined. This should come from the resolution user wants

scale : Scale factor to be applied after complex multiplication

@scratch This kernel needs following scratch buffers

@outputs This kernel produce following outputs

outputData

Pointer to the output buffer containing the output of this kernel which is stored with real and imaginary part interleaved .
Size of this buffer should be ((numDetections*numAngles) * size(int16_t) * 2)

@remarks Following is the buffer placement assumed for optimal performance of this kernel
inputData, outputData, steeringMatrix should all lie in different memory

@constraints Following constraints
Pitch, inputData and outputData should be word aligned

@return NONE

```
void vcop_beam_energy_calculation_kernel
(
 __vptr_int16    inputData,
 __vptr_uint32   outputEnergy,
 unsigned short  numDetections,
 unsigned short  numAngles
)
```

@desc This kernel does computes the energy after beam forming

@inputs This kernel takes following Inputs
inputData :
Input buffer containing data 16 bit signed data with real and imaginary part interleaved. Input buffer is of dimension numDetections x numAntennas, here detections are in horizontal direction and anetnna data is in vertical direction.
Size of this buffer should be numDetections* numAntennas * sizeof(int16_t) * 2
numDetections :
Number of detections whose angle needs to be determined
numAngles :
Number of angles to be determined. This should come from the resolution user wants

@scratch This kernel needs following scratch buffers

@outputs This kernel produce following outputs
outputEnergy
Pointer to the output buffer containing energy of the input data.
Size of this buffer should be ((numDetections * numAngles) * size(uint32_t))

@remarks Following is the buffer placement assumed for optimal performance of this kernel
inputData and outputData should be in two different buffers for best performance

@constraints Following constraints
numAngles should be even number

@return NONE

```
void vcop_beam_angle_association_kernel
(
  __vptr_uint32   inputEnergy,
  __vptr_uint16   angleBuf,
  __vptr_uint16   energyBuf,
  __vptr_uint16   ptrToInfoBuffer,
  __vptr_uint32   ptrToParamBlock,
  unsigned short  baseAngleOffset,
  unsigned short  numDetections,
  unsigned short  numAngles,
  unsigned short  energyScalingFactor
)
```

@desc This kernel associates an angle to each detection by finding max energy among all the angles

@inputs This kernel takes following Inputs

inputEnergy :
Pointer to the input buffer containing energy of the input data.
Size of this buffer should be ((numDetections * numAngles) * size(uint32_t))

baseAngleOffset :
Offset that need to be added for each angle

numDetections :
Number of detections whose angle needs to be determined

numAngles :
Number of angles to be determined. This should come from the resolution user wants

ptrToValidDetectionCount :
This field tells how many detections are actually valid in current iteration

ptrToParamBlock :
Pointer to the param block for this kernel. this will be used to update certain enteries in param block

energyScalingFactor:
Scale factor to apply (rounding) before storing the 32 bit enery in 16 bit container

@scratch This kernel needs following scratch buffers

@outputs This kernel produce following outputs

angleDetectionBuf
Pointer to the output buffer containing angle and detection id in interleaved manner
Size of this buffer should be ((numDetections * 2 * size(uint16_t))

energyBuf
Pointer to the output buffer containing energy for each detection
Size of this buffer should be ((numDetections * size(uint16_t))

```
    @remarks  This kernel invovles param block update and hence it is very important to
              note that if you make any changes to this kernel you should make sure
              that the corresponding param block offsets are updated accordingly

    @constraints Following constraints
                 None

    @return    NONE

   void vcop_range_doppler_energy_angle_mapping_kernel
   (
    __vptr_uint32  coordinateBufEnergy,
    __vptr_uint16   angleDetectionMapping,
    __vptr_uint16   angleBuf,
    __vptr_uint16   energyBuf,
    unsigned short  coordinateBufPitch,
    unsigned short  numDetections
   )

   @desc     This kernel updates the new angle and energy to the corresponding range and
             doppler coordinates in coordinate buffer. It is important to note that range
             and doppler dimension are expected to be present by default.
             Coordinate buffer is expected to be as follows :
              typedef struct
              {
                uint16_t      range;
                uint16_t      velocity;
                uint16_t      energy;
                uint16_t      angleBin;
               } BEAM_FORMING_TI_Coordinates;

    @inputs   This kernel takes following Inputs
              angleDetectionMapping :
                    Pointer to angle detection buffer which holds the mapping of a particular angle
with
                    correspondind detection number
                     Size of this buffer should be ( numDetections * sizeof(uint16_t) * 2)
              angleBuf :
                    Pointer to angle buffer which holds the angle at each detection
                     Size of this buffer should be ( numDetections * sizeof(uint16_t))
              energyBuf :
                    Pointer to energy buffer which holds the max energy of all angles at each
detection
                     Size of this buffer should be ( numDetections * sizeof(uint16_t))
              numDetections :
```

Number of detections whose angle needs to be determined
coordinateBufPitch :
Pitch in bytes to reach to the next detection coordinates from the first detection

@scratch This kernel needs following scratch buffers

@outputs This kernel produce following outputs
coordinateBufEnergy :
Pointer to coordinate buffer energy field as described in above description.
Size of this buffer should be (numDetections *
sizeof(BEAM_FORMING_TI_Coordinates))

@remarks Following is the buffer placement assumed for optimal performance of this kernel
coordinateBuf IBUFLA
angleDetectionBuf WBUF
energyBuf WBUF
@constraints Following constraints
None

@return NONE

5. Peak Detection

Description

These set of kernels are optimized implementation of various kernels required for peak detection.

Location

<EVE_SW_ROOT>/kernels/radarlib/vcop_beam_forming

Usage

These routines is C-callable and can be called as:

```
void vcop_tx_decoding_kernel
(
  __vptr_int16_arr inputData,
  __vptr_int8      txDecodingCoeff,
  __vptr_int16_arr outputData,
  unsigned short  numTx,
  unsigned short  numRx,
  unsigned short  numRows,
  unsigned short  numHorzPtPerAntenna,
  unsigned short  offsetBwTx,
  unsigned short  offsetBwRx,
  unsigned short  pitch
)
```

@desc This kernel does tx decoding by multiplying data received by all the receivers by a matrix of dimension numTx X numTx. This kernel has a foreach loop and each iteration works on 8 points at a time

@inputs This kernel takes following Inputs

inputData :

Input buffer containing data 16 bit signed data with real and imaginary part interleaved. This data is an array of pointers. for kth for each iteration the pointer in this array buffer should point to k * VCOP_SIMD_WIDTH Size of this buffer should be numTx * numRx * numHorzPtPerAntenna * sizeof(int16_t) * 2 * numRows

txDecodingCoeff :

Pointer storing the coefficient of tx decoding. Size of this buffer should be numTx * numTx

numTx :

Number of transmitters in the system

numRx :

Number of receivers in the system

numRows :

Number of rows to work with

numHorzPtPerAntenna :
Number of horizontal points in the input buffer per antenna
offsetBwTx :
Offset in bytes between two transmitter data
offsetBwRx :
Offset in bytes between two receiver data
pitch :
Offset in bytes to jump from one line to next line

@scratch This kernel needs following scratch buffers
None

@outputs This kernel produce following outputs
outputData
Output buffer containing data 16 bit signed data with real and imaginary part
interleaved. This data is an array of pointers. for kth for each iteration
the pointer in this array buffer should point to k * VCOP_SIMD_WIDTH
Size of this buffer should be numTx * numRx * numHorzPtPerAntenna *
sizeof(int16_t) * 2 * numRows

@remarks Following is the buffer placement assumed for optimal performance of this kernel
inputData : IBUFLA
outputData : IBUFHA
txDecodingCoeff : WMEM

@constraints Following constraints

@return NONE

```
void vcop_peak_detection_energy_across_antenna
(
  __vptr_int16    inputData,
  __vptr_uint32   outputData,
  unsigned short  numRows,
  unsigned short  numAntennas,
  unsigned short  numHorzPtPerAntenna,
  unsigned short  pitch
)
```

@desc This kernel computes the energy at each point and sum the energy
across all antenna for the same point

@inputs This kernel takes following Inputs
inputData :
Input buffer containing data 16 bit signed data with real and imaginary part
Interleaved. Input buffer is of dimension numAntennas x numDetections.

Size of this buffer should be numHorzPtPerAntenna *numRows * numAntennas * sizeof(int16_t) * 2

numRows :
Number of rows in the input block

numAntennas :
Total number of antennnas in the system

numHorzPtPerAntenna :
Number of horizontal points in the input buffer per antenna

pitch :
Offset in bytes to jump from one line to next line in input buffer

@scratch This kernel needs following scratch buffers
NONE

@outputs This kernel produce following outputs

outputData
Pointer to the output buffer containing the output energy of this kernel which is stored in uint32_t container.
Size of this buffer should be ((numRows * numHorzPtPerAntenna) * size(uint32_t))

@remarks Following is the buffer placement assumed for optimal performance of this kernel
inputData, outputData should be in different memory

@constraints Following constraints

@return NONE

```
void vcop_peak_detection_binlog_energy_scaling
(
  __vptr_uint32   inputEnergy,
  __vptr_uint16   indexBuf,
  __vptr_uint8    lmbdBuf,
  __vptr_uint16   lutTable,
  __vptr_uint16   lutValue,
  __vptr_uint16   scatterIndex,
  __vptr_uint16   outputEnergy,
  unsigned short  outputPitch,
  unsigned char   lutQFormat,
  unsigned char   alphaQFormat,
  unsigned short  numRows,
  unsigned short  numHorzPtPerAntenna
)
```

@desc This kernel computes computes the log2 of energy (32 bit) and store the output in 16 bit Container. Following is the explanation of this kernel

Binary Log can be computed as follows (log used here is log in base 2
Any number X can be written as 2^m (1 + aplha) where 0<= alpha < 1
log X = m + log (1 + aplha)
If n is precision for alpha, i.e. we have 2^n entries in lookup table to store log(1+aplha)
log X =~ m + log (1 + (floor(2^n * alpha)) / 2^n)
Index for LUT is in Qn format alpha = (X * 2^n/ 2^m) - 2^n
Which can also be written as X>> (m -n) - 2^n
Here we can avoid subtraction by 2^n if we store index as 1+alpha

@inputs This kernel takes following Inputs

inputEnergy :

Input buffer containing data 16 bit signed data with real and imaginary part interleaved. Input buffer is of dimension numAntennas x numDetections.

Size of this buffer should be numHorzPtPerAntenna *numRows * numAntennas * sizeof(int16_t) * 2

lutTable :

LUT table to store the LUT of log (1 + (floor(2^n * alpha)) / 2^n). LUT values are stored in Q format which is given by lutQFormat parameter. The index of LUT will be given by 1+alpha.

Size of this buffer should be 2 * (1<< alphaQFormat). Multiplication by 2 because we are allocating double the size of LUT so that we can avoid subtraction by 2^n

scatterIndex :

Pointer to the index to be used for pScatter. This should be such that all 8 lanes of a vector goes to different memory bank.

size of this buffer should be VCOP_SIMD_WIDTH * sizeof(uint16_t)

outputPitch :

Pitch in bytes used to store the output as transpose. Pitch should be chosen in a way that it is greater than 9 (in terms of number of words) and is an odd number

lutQFormat :

Q Format to be used to store the LUT table

alphaQFormat :

Q Format to be used finding the index corresponding to LUT table

numRows :

Number of rows in the input block

numHorzPtPerAntenna :

Number of horizontal points in the input buffer per antenna

@scratch This kernel needs following scratch buffers

indexBuf

This is a scratch buf used to store the index for LUT table

lutValue

This is a scratch buf used to store the LUT value after reading from LUT table

@outputs This kernel produce following outputs

outputData

Pointer to the output buffer containing the output energy of this kernel which is stored in uint32_t container.

Size of this buffer should be ((numRows * numHorzPtPerAntenna) * size(uint32_t))

@remarks Following is the buffer placement assumed for optimal performance of this kernel
inputEnergy : IBUFLA
indexBuf : IBUFLA
lmbdBuf : WMEM
lutTable : WMEM
scatterIndex : WMEM
outputEnergy : IBUFHA

@constraints Following constraints
numHorzPtPerAntenna should be multiple of VCOP_SIMD_WIDTH

@return NONE

```
void vcop_peak_detection_cell_sum
(
  __vptr_uint16   inputEnergy1,
  __vptr_uint16   inputEnergy2,
  __vptr_uint16   cellSum,
  __vptr_uint32   cellSumOneLine,
  unsigned short  noiseLen,
  unsigned short  gaurdLen,
  unsigned short  numHorzPoint,
  unsigned short  numVertPoint,
  unsigned short  lineOffsetInBytes,
  unsigned short  shift
)
```

@desc This kernel computes the cell sum using the energy buffer

@inputs This kernel takes following Inputs
inputEnergy1 :
Input buffer containing data 16 bit energy of first (numVertPoint/2) rows
Size of this buffer should be numHorzPoint *(numVertPoint /2) * sizeof(uint16_t)
inputEnergy2 :
Input buffer containing data 16 bit energy of (numVertPoint/2 + 1) to numVertPoint rows.
Size of this buffer should be numHorzPoint *(numVertPoint /2) * sizeof(uint16_t)
noiseLen :
Noise length to be used to calculate the nosie floor
gaurdLen :
gaurd length to be used to to skip gaurd cells
numHorzPoint :
Number of horizontal points in the input block
numVertPoint :

Number of rows in the input block
lineOffsetInBytes :
Offset in bytes to jump from one line to other line
shift :
Shift to be applied to the output

@scratch This kernel needs following scratch buffers
NONE

@outputs This kernel produce following outputs
cellSum
Pointer to the output buffer containing the cell sum of this kernel which is stored in uint16_t container.
Size of this buffer should be (((numVertPoint + 2 * (noiseLen + gaurdLen)) * numHorzPtPerAntenna) * size(uint16_t))
cellSumOneLine
This buffer stores a copy of cell sum for initial one line for both energy1 and energy2
buffers without doing any saturation.
Size of this buffer should be ((numHorzPtPerAntenna) * size(uint32_t) * 2)

@remarks Following is the buffer placement assumed for optimal performance of this kernel
inputEnergy1 : IBUFLA
inputEnergy2 : IBUFHA
cellSum : WMEM
cellSumOneLine : IBUFHA

@constraints Following constraints
numHorzPoint should be multiple of VCOP_SIMD_WIDTH

@return NONE

```
void vcop_peak_detection_CFARCA_thresholding
(
  __vptr_uint16   inputEnergy,
  __vptr_uint16   cellSum,
  __vptr_uint8    binaryMask,
  __vptr_uint32   outRangeDopplerBuf,
  __vptr_uint16   outEnergyBuf,
  __vptr_uint32   idxBuf,/* Store index left shifted by 16*/
  __vptr_uint32   dopplerIdxOffsetBuf,
  __vptr_uint16   numDetections,
  unsigned int    offsetBwTwoInBuf,
  unsigned short   noiseLen,
  unsigned short   gaurdLen,
  unsigned short   numHorzPointActual,
```

```
    unsigned short   numHorzPoint,
    unsigned short   numVertPointActual,
    unsigned short   numVertPoint,
    unsigned short   lineOffsetInBytesActual,
    unsigned short   lineOffsetInBytes,
    signed short   threshold

)
```

@desc This kernel does CFAR CA detector and store the output coordinates

@inputs This kernel takes following Inputs

inputEnergy :
Input energy buffer containing 16 bit energy
Size of this buffer should be numHorzPoint *numVertPoint * sizeof(uint16_t)

cellSum :
Buffer which stores the cell sum computed byt the previous kernel
Size of this buffer should be (((numVertPoint + 2 * (noiseLen + gaurdLen)) * numHorzPtPerAntenna) * size(uint16_t))

idxBuf :
Buffer to store the index of doppler dimension.Refer testbench to see how it is computed
Size of this buffer should be (numHorzPtPerAntenna * 2 * size(uint16_t))

dopplerIdxOffsetBuf :
Offset to be added to the doppler index
Size of this buffer should be sizeof(uint32_t))

numDetections :
Buffer to store the number of detections
Size of this buffer should be (VCOP_SIMD_WIDTH * 2 * size(uint16_t))

offsetBwTwoInBuf :
Offset in bytes between inputEnergy1 and inputEnergy2 as used in vcop_peak_detection_cell_sum kernel

noiseLen :
Noise length to be used to calculate the nosie floor

gaurdLen :
gaurd length to be used to to skip gaurd cells

numHorzPointActual :
Number of actual horizontal points in the input block

numHorzPoint :
Number of horizontal points in the input block. when numHorzPointActual == 8 then numHorzPoint = 16 rest all cases numHorzPoint = 8

numVertPointActual :
Number of actual vertical in the input block.

numVertPoint :
Number of vertical in the input block. When numHorzPointActual == 8

then numVertPoint = numVertPointActual / 2 rest all cases numVertPoint = numVertPointActual

lineOffsetInBytesActual :

Actual Offset in bytes to jump from one line to other line

lineOffsetInBytes :

When numHorzPointActual == 8.

then lineOffsetInBytes = lineOffsetInBytes *2 rest all cases lineOffsetInBytes = lineOffsetInBytes

threshold:

Threshold to be used for CFAR CA DB. Original equation of CFAR CA detector is CUT > T * NF, where T can be written as C1/ 2^C2

Same equation in log domain becomes

log(CUT)> log(T) + log(NF) = log(CUT) - log(NF)> log(C1) - C2

So threshold here is log(C1) - C2.

Q format for threshold should be same as lutQFormat used during energy computation

@scratch This kernel needs following scratch buffers

binaryMask

Buffer to store the binary mask

Size of this buffer should be numHorzPoint * numVertPoint

@outputs This kernel produce following outputs

outRangeDopplerBuf

Pointer to the output buffer storing the range and doppler coordinate for each detection

outEnergyBuf

Pointer to the output buffer storing the energy for each detection

@remarks Following is the buffer placement assumed for optimal performance of this kernel

inputEnergy : IBUFLA and IBUFHA

cellSum : WMEM

binaryMask : WMEM

outRangeDopplerBuf : IBUFLA

outEnergyBuf: IBUFHA

dopplerIdxOffsetBuf : WMEM

numDetections : IBUFHA

@constraints Following constraints

numHorzPoint should be multiple of VCOP_SIMD_WIDTH (an power of 2.

@return NONE