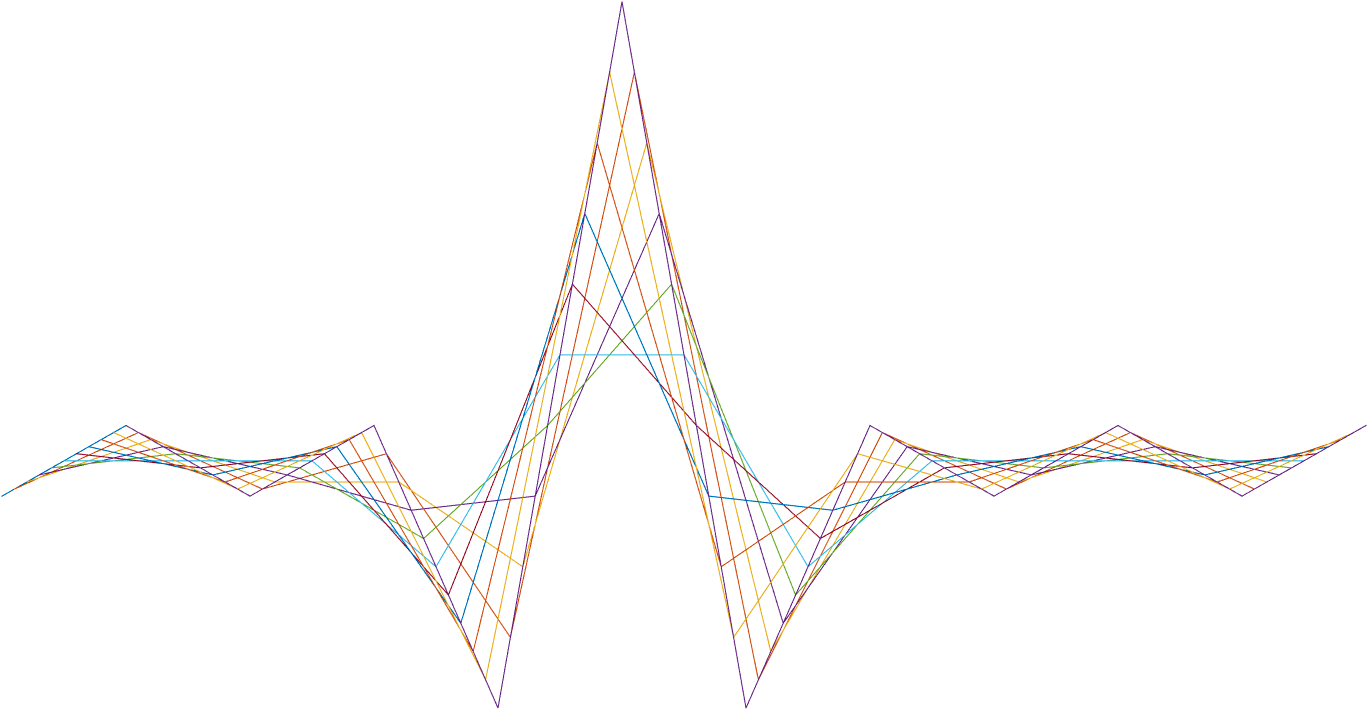
IVCAM2.0 3D Imaging Camera



ASIC A0 Rasterization specification

9 February 2017

Revision 0.9

Intel Top Secret

Table : Revision history

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Matlab Version | Revision Number | Revised by | Description | Revision Date |
| TBD | 0.3.0 | Ohad Menashe | Initial Release | 03 Feb 2016 |
| TBD | 0.3.1 | Vitaly Surazhsky | PCQ seriously revised | 21 Feb 2016 |
| TBD | 0.3.2 | Ohad Menashe | IR partially added, overall revisiting | 31 March 2016 |
| TBD | 0.5.0 | Vitaly Surazhsky | IR added, filter seriously revised | 19 April 2016 |
| TBD | 0.5.1 | Vitaly Surazhsky | Minor fixes | 2 May 2016 |
| TBD | 0.5.2 | Vitaly Surazhsky | IR adaptor block added, confDC added | 4 May 2016 |
| 2016a | 0.5.3 | Vitaly Surazhsky | IR adaptor block removed, minor fixes | 8 May 2016 |
| 2016a | 0.6.0 | Vitaly Surazhsky | Support for non 2^N codes, flags | 7 June 2016 |
| 2016a | 0.6.1 | Vitaly Surazhsky | *ld\_on* flag and end of frame handling added | 14 June 2016 |
| 2016a | 0.6.2 | Vitaly Surazhsky | CMA Filter block modified to meet ASIC design | 28 July 2016 |
| 2016a | 0.6.5 | Vitaly Surazhsky | CMA Filter weights computation complete | 25 Sep 2016 |
| 2016a | 0.7.0 | Vitaly Surazhsky | CMA Collector IR variance, test plan added | 26 Sep 2016 |
| 2061a | 0.7.1 | Vitaly Surazhsky | input/output flags are finalized, added RegsRASTskipOnTxModeChange | 6 Oct 2016 |
| 2061a | 0.7.2 | Vitaly Surazhsky | input flags order of bit 0 and 1 fixed | 9 Oct 2016 |
| 0.7.42 | 0.7.3 | Ohad Menashe | Interface alignment | 13 Oct 2016 |
| 0.7.43 | 0.7.4 | Vitaly Surazhsky | RegsRASTsideLobeDir added  Flags added to PCQ Pixel fifo, CMAC input | 19 Oct 2016 |
| 0.7.45 | 0.7.4 | Vitaly Surazhsky | PCQ: pixel x coordinated is computed by rounding (floor previously), see Section 3.1.1 | 26 Oct 2016 |
| 0.7.45 | 0.7.4 | Vitaly Surazhsky | RegsRASTMaxLdOffChunks removed; inRoi is computed and used intead of ld\_on; Size of IR, IRSq and IR counter reduced by 6 bits; PCQ: IR values of side lobes chunks are zero; | 2 Nov 2016 |
| 0.7.45 | 0.7.5 | Vitaly Surazhsky | Section : conf DC computation is updated | 8 Nov 2016 |
| 0.7.45 | 0.7.6 | Vitaly Surazhsky | TxRx = 3: Sections 3.1.1, 3.1.2, 3.1.4 | 16 Nov 2016 |
| 0.8.49 | 0.8.1 | Vitaly Surazhsky | RegsRASTinvalidateDiffTxRx added in 3.1.2 | 30 Nov 2016 |
| 0.9 | 0.9 | Vitaly Surazhsky | RegsRASTdiscardLateChuncks added in 3.1.2 | 4 Jan 2017 |

2

Contents

[1 Introduction 4](#_Toc467166971)

[2 Interfaces 6](#_Toc467166972)

[2.1.1 Input 6](#_Toc467166973)

[2.1.2 Output 7](#_Toc467166974)

[3 Detailed description 8](#_Toc467166975)

[3.1.1 Pixel Chunk Quantizer (PCQ) 8](#_Toc467166976)

[3.1.2 CMAC: Code measurement accumulator collector 11](#_Toc467166977)

[3.1.3 CMA Filter 12](#_Toc467166978)

[3.1.4 Invalid pixels with TxRx = 3 14](#_Toc467166979)

[3.1.5 CMA Filter Bypass 14](#_Toc467166980)

[3.1.6 Confidence DC computation 14](#_Toc467166981)

[4 LUTs 15](#_Toc467166982)

[5 Registers 16](#_Toc467166983)

[6 Memory and computation operations 18](#_Toc467166984)

[7 Open Issues 19](#_Toc467166985)

[7.1 Status registers 19](#_Toc467166986)

[7.2 Flag implementation description 19](#_Toc467166987)

[8 Test plan 20](#_Toc467166988)

Tables

[Table 1: Revision history 1](#_Toc450467544)

[Table 2: Registers 10](#_Toc450467545)

[Table 3: DRAS memory and bandwidths 11](#_Toc450467546)

Figures

[Figure 1: RAST block diagram 3](#_Toc450467537)

[Figure 2: PCQ 5](#_Toc450467538)

[Figure 3: Chunk FIFO and Pixel FIFO with its elements pointing to chunk locations of Chunk FIFO. 6](#_Toc450467539)

[Figure 4: CMA element structure 7](#_Toc450467540)

[Figure 5: CMAC 7](#_Toc450467541)

[Figure 6: CMA Filter 8](#_Toc450467542)

Introduction



The RAST block is the first to process the raw input data. It is divided into three main parts: the pixel chunk quantizer (PCQ), the code measurement accumulator collector (CMAC) and CMA filter.

The PCQ goal is to synchronize the input to the spatial pixel grid and make sure that every scan cross of a pixel has full code length. The input fast channel data is divided into chunks of 64 sample bits for efficient handling and storage. This quantizing was evaluated to not degrade the image location and data quality. Every chunk of the sample data comes with its IR data and its (*x*, *y*) pixel location.

Since the pixel time varies throughout the scan line, the number of chunks per pixel varies as well. In case where the number of chunks per pixel is shorter than the sampled code length *K/64*, the pixel data is augmented by padding using chunks from neighboring pixels. As a result, the PCQ number of output chunks can be greater than the number of PCQ input chunks (i.e.; the throughput), when some chunks can be assigned or used by up to three pixels. So the PCQ output throughput rate can be three times the input sampling rate *FS*.

CMAC accumulates all pixel chunks that correspond to the same pixel within CMAs. Every CMA is a container that holds *K* bit accumulators, and K/64 sample counters, IR accumulator and IR counter. The sample and IR counters are incremented when CMAC aggregates chunks. CMAC outputs a CMA of a pixel when all scanlines have visited the pixel.

CMA filter normalizes the CMA and IR values using the corresponding counters and adaptively averages the resulting CMAs within a 3x3 window. The weights of the adaptive averaging are based on IR values.



Figure : RAST block diagram

The RAS output is a stream of normalized CMA (without counters) pixel structures. This structure is outputted directly to DCOR (correlation) processing along with its IR and *(x, y)* pixel location.

Interfaces

The input to RAST is coming from the analog fast channel at the configured sampling rate as a 1-bit data stream, 12-bit IR data and location information for each chunk of input data.

Transmission (TX) rates are 1Gsps, 833Msps 500Msps, and 250Msps.

The fast channel sampling rate and transmission (TX) rate are related in a ratio of 1:4, 1:8, and 1:16.

The slow channel sampling rate is with a ratio of 1:8, 1:4 and 1:2 to the TX rate.

The supported code lengths are any multiplication of 2, between 8 and 128.

Let be the sampled code length:  
*N* is the code length; even numbers in [8..128] are supported.  
*FS* is the sampling ratio, 1:4, 1:8, and 1:16.  
Tx is the symbol transmission time – we support 1/2/4nsec.  
For example: K=1024 = 64(N) \* 16G(Fs) \* 1nsec(Tx).

Geometric Engine (GE) provides positions every 64 bit of the fast channel data. The maximal supported resolution is 4096 x 2048. GE generates x-coordinate with 15 bit accuracy (12 bit, max 4096 + 2 bit extra precision + 1 bit sign) and y-coordinate with 12 bit accuracy (11bit, max 2048 + 1 bit sign). PI and GE support various cropping modes when the resulting image is generated only for a portion of the entire FOV. In these cropping modes, GE generates positions that already include cropping correction. This way, valid pixel positions of the resulting output image are always between (0, 0) and (MaxXRes – 1, MaxYRes – 1).

The code lengths and sampling rates together with image resolutions are constrained to get rid of inefficient configurations. These constraints should save ASIC area resources, improve performance and ease implementation.

Input

Chunks containing the following data:

* fast\_channel: 64 binary samples, 64 bit:
* slow\_channel: IR data:
  + 12bit every 64 bits of fast channel
* noise\_est: IR noise estimation:
  + 12bit every 64 bits of fast channel
* loc\_xy: chunk positions, every 64 bit of fast channel data:
  + X coordinate: 15 bit (12 bit, max X-Res 4096 + 2 bit extra precision + 1 bit sign)
  + Y coordinate: 12 bit (11 bit, max Y-Res 2048 + 1 bit sign)
* flags (pi\_flags + roi), 6 bits:
  + [0] ld\_on: laser on, 1 bit every 64 bits of fast channel
  + [1] tx\_code\_start: code start flag to indicate that the chunk is aligned with the start of the transmitted code (at least once per scanline), 1 bit every 64 bits of fast channel
  + [2] scan\_dir: 1 bit, scanline direction
  + [4:3] txrx\_mode: 2 bits, transmission mode
  + [6] roi: 1 bit, regions of interest, includes the entire image, raise is used to make the start of frame, fall is used to mark the end of frame

Output

Pixel normalized CMA, see Table 3:

* Samples: K \* 8 bit
* IR: 12 bit
* noise\_est: 12 bit
* raster position: 12 bit + 11 bit
* conf\_dc: 4 bit
* pxl\_flags: 4 bits
  + [0] scan\_dir: 1 bit, scanline direction
  + [2:1] txrx\_mode: 2 bits, transmission mode
  + [3] eof: 1 bit, end of frame

Detailed description

Pixel Chunk Quantizer (PCQ)



Figure : PCQ

The implementation of PCQ consists of a chunk FIFO buffer (Chunk FIFO), pixel FIFO buffer (Pixel FIFO), and two processing blocks, Analyzer and Writer.

Chunk FIFO contains roughly of the chunks of a single scanline. In the worst case with sampling rate of 16Gb/s, the size of the chunk FIFO should be:

The mirror nominal frequency is 20K. There mirror can vary up to 7.5% from nominal.

The dead time where the laser is not projecting (since the mirror movement is too slow) is 1-1.5deg, which at 1deg is leading to actual 76% of the scan duty cycle. Note that the buffer size of power of 2 easies the implementation and we use 8192 for Chunk FIFO size.

Every Chunk FIFO element consist of:

* samples: [64 bits] the input samples,
* IR: [12 bits] the IR sample data,
* NEST: [12 bits] the IR sample data,
* offset: [8 bits] offset of the chunk data within the code.
* txrx\_mode: [2 bits] txrxr\_mode of the chunk

Pixel FIFO maintains the information on how the chunks of Chunk FIFO should be accessed and released by Writer. The size of Pixel FIFO is the size of the scanline. Each Pixel FIFO element contains:

* position: [12 + 11 bits] (*x, y*) pixel position,
* copyStart, copyEnd: [2x13 bits] indices of the start and end positions of the pixel chunks within Chunk FIFO that Writer outputs,
* start, end: [2x13 bits] indices of the start and end positions of the chunks within Chunk FIFO that Writer uses to release chunks from Chunk FIFO when they are no longer needed.
* pxl\_flags: 4 bits
  + [0] scan\_dir: 1 bit, scanline direction
  + [2:1] txrx\_mode: 2 bits, transmission mode

Analyzer is the block that pushes the input data into Chunk FIFO and generates Pixel FIFO elements for every pixel in a scan-line. It guarantees that every pixel has enough chunks to cover the entire code length. Analyzer puts every chunk that has *~~ld\_on~~* ~~flag~~ *inRoi* value set into Chunk FIFO. *inRoi* is true iff –*MaxSideLobe* <= y && y <= RegsGNRLimgVsize + *MaxSideLobe*, when *MaxSideLobe* is 1+2\* txrx\_mode of the chunk. Every time it puts a chunk with a different than previous y-position meaning that a chunk of the new pixel has been started, it also puts a new element into Pixel FIFO for the previous pixel with the indices of its chunks and its pixel position. The *xy*-position of the Pixel FIFO element is computed as the average of the first and the last chunks of the pixels. The extra 2 bits of x-position is then removed by rounding (bitShiftRight(*xfirst* + *xlast* + 4, 3)) and only the signed values (12 + 11 bits) of *xy-*position are further used. Note that *xy*-position of the chunks are not stored in Chunk FIFO. *xy*-position will be assigned to the chunks by Writer using the xy-position from the Pixel FIFO element.

copyStart, copyEnd are computed such that the number of chunks per pixel is at least to cover the entire sampled template. RegsRASTsideLobeDir defines whether we add the same number of chunks before and after the pixel. If RegsRASTsideLobeDir is not set and the number of chunks to add is odd, we add more chunks before the pixel chunks.

Note: To reduce complexity, the chunks of a single scanline are sub-divided into pixels according to their vertical y-positions only. This way every scanline produces exactly RegsGNRLimgVsize pixels.

We assume that every chunk can be assigned to at most 3 pixels. Thus the chunks must be pulled out from Chunk FIFO by Writer at a rate three (3) times the input rate. The amortized rate could be less than 3, approximately 2.4, due to varying speed of mirror movement. The correct Writer output rate and Chunk FIFO size must ensure that Chunk FIFO does not overflow. The number of chunks in Chunk FIFO should be at least a single scanline, which is chunks to guarantee constant output rate regardless of the non-constant pixel time for the optimal output rate (2.4).

Writer set output IR values of chunks to 0 for all the chunks that are in the side lobe regions, namely, not between start and end pointers.

Writer does not output chunks whose txrx\_mode is different from the txrx\_mode of the pixel.



Figure : Chunk FIFO and Pixel FIFO with its elements pointing to chunk locations of Chunk FIFO.

The PCQ output and CMAC input interface:

* position: [12 + 11 bits] (*x, y*) pixel position
* samples: [64 bits] the input samples,
* IR: [12 bits] the IR sample data,
* NEST: [12 bits] the IR sample data,
* offset: [8 bits] offset of the chunk data within the code
* pxl\_flags: 4 bits
  + [0] scan\_dir: 1 bit, scanline direction
  + [2:1] txrx\_mode: 2 bits, transmission mode
  + [3] eof: 1 bit, end of frame, indicates the last (dummy) pixel of the frame

CMAC: Code measurement accumulator collector



Figure : CMA element structure

The CMA is the pixel representation of all of the binary input data that assigned to a pixel. Every CMA has *K* (the length of the sampled code length) elements that store accumulated code, while every element is the summation of the accumulated samples, and the counter (denominator) of the number of samples. The counter (denominator) is shared between 64, 32, 16 or 8 samples. Depending on the code size and sampling rate. The Register parameter that dictates this is RegsRASTSharedDenom.

The number of bits (from 2 to 7) of the CMA bin element varies with the configurations (i.e. sampling rate, transmitted code length, and number of scan-lines per pixel). For example, if the entire code appears 7 times in a single pixel, the CMA element should have at least 3 bits. The CMA element structures also contains IR accumulator (12+10 bits), IRSq accumulator (12+12+10 bits), IR counter (10 bits) and NEST values (12 bits). IRSq accumulates squares of IR values for estimation of IR variance. The accumulation of IR and IRSq is stopped when IR counter reaches its maximal value (1023).



Figure : CMAC

The CMAC receives chunks from PCQ, accumulates them within CMAs and outputs CMAs to CMA filter. The CMAC is a data structure that consists of a bank of CMA, for a single raster column (RegsGNRLimgVsize), and several fast CMAs where summation takes place. All the CMAs are stored together with their pixel *xy*-position.

We assume that chunks from PCQ come in a specific order based on the mirror scan-lines. Thus, using the knowledge that the *xy* positions of chunks change monotonously, we can preload a (soon-to-be-needed) CMA from the CMA bank into a fast CMA, and save it back into the CMA bank, while a current chunk is being aggregated into the current fast CMA. The preloading and saving of fast CMAs are based on the value of *y-*coordinate (fast scan) of the current chunk. A CMA leaves the CMA bank when the current input chunk has its *x*-coordinate different from that of the preloaded fast CMA with the same *y*-position. This allows us to store only one CMA for every raster row in the CMA bank regardless the scan-line distortion.

If x-coordinate of the chunk is less than x-1 of the current CMA bank, namely, the chunk arrived later relatively to its x-position than the previous chunks of the same row, the chunk is discarded if RegsRASTdiscardLateChuncks is set and accumulated as usual otherwise.

When a chunk is aggregated into a CMA its rxtx\_mode is compared against that of the CMA. If the rxtx\_mode of the CMA is not set (NEST = 0), the CMA rxtx\_mode bits are updated with that of the chunk. Otherwise (NEST != 0), the CMA rxtx\_mode is set to 3 (invalid) if the the rxtx\_mode value of the chunk is different from the CMA value and RegsRASTinvalidateDiffTxRx is set.

When a CMA leaves the CMAC block its IR and NEST values are being normalized by dividing the IR and NEST samples sum by their corresponding counters. If RegsRASToutIRvar is set, the IR variance of the pixel is computed as follows: IRsq/IRcounter – (IR/IRcounter)^2. The divisions are done in single precision floating point and the resulting values are converted to 12 bit unsigned integer.

CMA Filter



Figure : CMA Filter

Since CMAs leave CMA bank at different rates depending on the fast mirror speed, CMAs are written into the buffer called CMA FIFOs. CMA FIFOs is the three columns of the maximal Y resolution (960). CMA FIFOs allow to perform bilateral 3 x 3 smoothing filtering of CMAs.

When a CMA enters CMA Filter, it is pushed into one CMA buffer called Entry CMA. BL filter then processes the 3x3 window of the CMA FIFOs that corresponds to Entry CMA y-coordinate. The current CMA of row *y* is advanced to the next CMA according the FIFO order of the row. After the 3x3 window of row *y* has been processed, the CMA from Entry CMA buffer is moved into the CMA FIFO of row *y.*

Not every 3x3 window can be processed by BL Filter, since in the beginning not all the necessary pixel of every 3x3 window have arrived. To be able to decide whether a 3x3 window can be processed by BL Filter, we use a counter for every CMA FIFOs row.

In the beginning of the frame, the counters are initialized with value 2. When the CMA from Entry CMA buffer pushed into its row of CMA FIFOs, we decrease the row counter*.* When BL Filter processes row *y*, it first checks that the counter of the row *y* is zero. Only in this case BL Filter processes the 3x3 windows of row *y* and outputs smoothed normalized CMA.

BL Filter block (bilateral filter) takes the current CMA from CMA FIFOs, normalizes them and fills a 3x3 window of normalized CMAs. Note that not all the neighbors can be always located.

Normalization of the CMA values is performed by division by their corresponding counters (denominators). The sample division is performed using a LUT of 6 bits to 8 bits.

Weighted averaging performed by BL Filter is adaptive and is based on the normalized IR values.

The resulting CMA is computed using the current CMA and its located neighbors as following:

First, we compute a robust estimation, *IR,* of the normalized *IR* values (intensities) using mean-median (MM). MM of the normalized IR values is computed by sorting the IR values and taking weighted average of the three middle elements. The weights of averaging are RegsRASTMMSide for the side elements, and 16 – 2\* RegsRASTMMSide for the central element.

Spatial weights *wsi* are defined as three different weights, one for the central pixel, one for the side pixels, and one for the diagonal pixels. The weights of the side pixels *wsside* are defined as follows:

|  |  |
| --- | --- |
| Bit0 of RegsRASTbiltAdapt | Computation of *wsside* |
| 0 | LUTRASTbiltSpat[min(63, bitshiftRight(16 \* RegsRASTbiltSharpnessS),4)] |
| 1 | LUTRASTbiltSpat[min(63, bitshiftRight(bitshiftRight(*IR*, 6) \* RegsRASTbiltSharpnessS),4)] |

The diagonal pixel weights, *wsdiag* are then defined as *wsdiag* = bitshiftRight(*wsside*\*RegsRASTbiltDiag, 4). The central pixel weight is defined as *wscentral* = 255 – 4\*(*wsside* + *wsdiag*).

The radiometric weights are computed depending on Bit1 of RegsRASTbiltAdapt as defined in the table below:

|  |  |
| --- | --- |
| Bit1 of RegsRASTbiltAdapt | Computation of |
| 0 | LUTRASTbiltSigmoid [min(63, min(1023,| *IR* – *IRi* |) \* LUTRASTbiltAdaptR[0] \* RegsRASTbiltSharpnessR),11)] |
| 1 | LUTRASTbiltSigmoid [min(63, bitshiftRight (min(1023,| *IR* – *IRi* |) \* LUTRASTbiltAdaptR[bitshiftRight(*IR*, 6)] \* RegsRASTbiltSharpnessR),11)] |

After the weights for pixels are computed we normalize them so that their sum will be 256. For that we compute *sfactor* = uint28(float32(2^(8+20))/float32()). If is equal to 0, *sfactor* is set to 0. The weights *wi =*for all the pixels except for the central pixel are defined as *wi =* bitshiftright*(wsi \* wri \* sfactor + 0, 20).* Theweight *wi*of the central is defined as . If is 0, CMA output is the same as the CMA of the central pixel.

Invalid pixels with TxRx = 3

Pixels with flags TxRx = 3 are considered invalid and are not used in smoothing computations of their neighbors, namely, their averaging weights are 0. Both depth weighted averaging and MM of IR values are not computed and the depth and IR output values are 0.

CMA Filter Bypass

The bypass mode controlled by RegsRASTSmoothBypass disables only the smoothing itself of CMAs. In Bypass mode CMA Fifo and FIFO Refs are not used, CMA smoothing and IR robust estimation are not computed. Normalization of the CMA is still performed, so that the output consists of normalized 8-bit CMA values.

Confidence DC computation

CMA Filter computes and outputs conf\_dc, which is based on the ratio of 0’s and 1’s in the original samples. To compute conf\_dc normalized CMA values are used as follows. First we compute the   
 LUTRASTconfDC(min(*|*RegsRASTdcLeve0X - *bitShiftRight*(*SCMA\** RASTdcCodeNorm,22)|*,63*)), where SCMA is the 20-bit sum of all the normalized CMA values of the pixel. RegsRASTdcLeve0X is one of the registers RegsRASTdcLeve00,.., RegsRASTdcLeve02 that is selected by the txRxMode of the pixel.

LUTs

Table : LUTs

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Name | Entries | Entry size | Fixed | Description |
| LUTRASTdivCma | 2^7 | 8 | yes | Division for CMA normalization |
| LUTRASTbiltSpat | 2^6 | 5 | no | Spatial weights |
| LUTRASTbiltAdaptR | 2^6 | 8 | no | Adaptive radiometric weights |
| LUTRASRbiltSigmoid | 2^6 | 8 | no | Sigmoid for radiometric weights |
| LUTRASTconfDC | 2^6 | 4 | no | confDC computation |

Registers

Table : Registers

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Name** | **Size** | **Default** | **Range** | **Special values/ description** |
| **General** |  |  |  |  |
| RegsGNRLimgHsize | 12 | 640 | 1-1280 | Horizontal resolution |
| RegsGNRLimgVsize | 12 | 480 | 1-960 | Vertical resolution |
| RegsGNRLcodeLength | 8 | 26 | 8, 16, 32, 64, 128 | The code length |
| RegsGNRLsampleRate | 5 | 16 | 4, 8, 16 | Sampling rate |
| **RAST** |  |  |  |  |
| RegsRASTcmaBinSize | 3 | 3 | 1-7 | The number of bits in CMA cells |
| RegsRASTsharedDenom | 7 | 64 | 8..64 | The number of bits shared by CMA denominator.  2^RegsRASTsharedDenomExp |
| RegsRASTsharedDenomExp | 3 | 6 | 3..6 | The exponent of RegsRASTsharedDenom |
| RegsRASTcmaMaxSamples | 7 | 7 | 1-127 | 2^ RegsDrCmaBinSize-1 |
| RegsRASTsideLobeDir | 1 | 1 | 0, 1 | 0: Larger side lobe before pixel wrt scan dir  1: Larger side lobe after pixel wrt scan dir |
| RegsRASTbiltBypass | 1 | 0 | 0, 1 | The smoothing is bypassed in CMA Filter |
| RegsRASTbiltAdapt | 2 | 3 | 0-3 | Bit 0: spatial adaptive Bit 1: radiometric adaptive |
| RegsRASTbiltSharpnessS | 6 | 16 | 0..63 | Spatial sharpness |
| RegsRASTbiltSharpnessR | 6 | 16 | 0..63 | Radiometric sharpness |
| RegsRASTbiltDiag | 5 | 16 | 0-16 | Bilateral spatial diagonal weight 16: the same weight as the side weight <16: a fraction of the side weight |
| RegsRASTmmSide | 3 | 4 | 0-7 | Side weight of the robust IR estimation using MM |
| RegsRASToutIRmm | 1 | 1 | 0, 1 | Robust estimation (MM) of IR replaces original IR |
| RegsRASToutIRvar | 1 | 0 | 0, 1 | IR variance replaces original IR |
| RegsRASTdcLeve\_000 RegsRASTdcLeve\_001 RegsRASTdcLeve\_002 | 8 | 121 | 32..223 | DC level for confidence DC computation |
| RASTdcCodeNorm | 16 | 1 | 2048..32768 | DC level normalization factor (div by K) |
| RegsRASTskipOnTxModeChange | 6 | 4 | 0..63 | The number of chunks to skip when TxRx mode is changing |
| RegsRASTinvalidateDiffTxRx | 1 | 0 | 0, 1 | Invalidate pixels with different TxRx mode |
| RegsRASTdiscardLateChuncks | 1 | 1 | 0,1 | Discard chunks with x smaller that the current chunk |

Memory and computation operations



Table : DRAS memory and computations

Open Issues

Status registers

Consider adding status registers that report various statistics and errors, e.g. FIFO overflows.

Flag implementation description

Test plan

|  |  |  |
| --- | --- | --- |
| **Name** | **Values** | **Distribution** |
| RegsRASTbiltBypass | 0 | 95 |
| 1 | 5 |
| RegsRASTbiltAdapt | 0 | 20 |
| 1 | 25 |
| 2 | 25 |
| 3 | 30 |
| RegsRASTbiltSharpnessS RegsRASTbiltSharpnessR | 0 | 5 |
| 1..15 | 30 |
| 16 | 40 |
| 17..63 | 25 |
| RegsRASTbiltDiag | 0..10 | 30 |
| 10..15 | 40 |
| 16 | 30 |
| RegsRASTmmSide | 0..3 | 40 |
| 4 | 40 |
| 5..7 | 20 |
| RegsRASToutIRmm | 0 | 40 |
| 1 | 60 |
| RegsRASToutIRvar | 0 | 85 |
| 1 | 15 |
| RegsRASTsideLobeDir | 0 | 50 |
| 1 | 50 |
| RegsRASTinvalidateDiffTxRx | 0 | 50 |
| 1 | 50 |
| RegsRASTdiscardLateChuncks | 0 | 50 |
| 1 | 50 |
| RegsRASTdcLeve\_000  RegsRASTdcLeve\_001  RegsRASTdcLeve\_002 | 0..115 | 10 |
| 116..132 | 80 |
| 133..255 | 10 |
| RegsRASTskipOnTxModeChange | 0 | 40 |
| 1..8 | 40 |
| 8..63 | 10 |