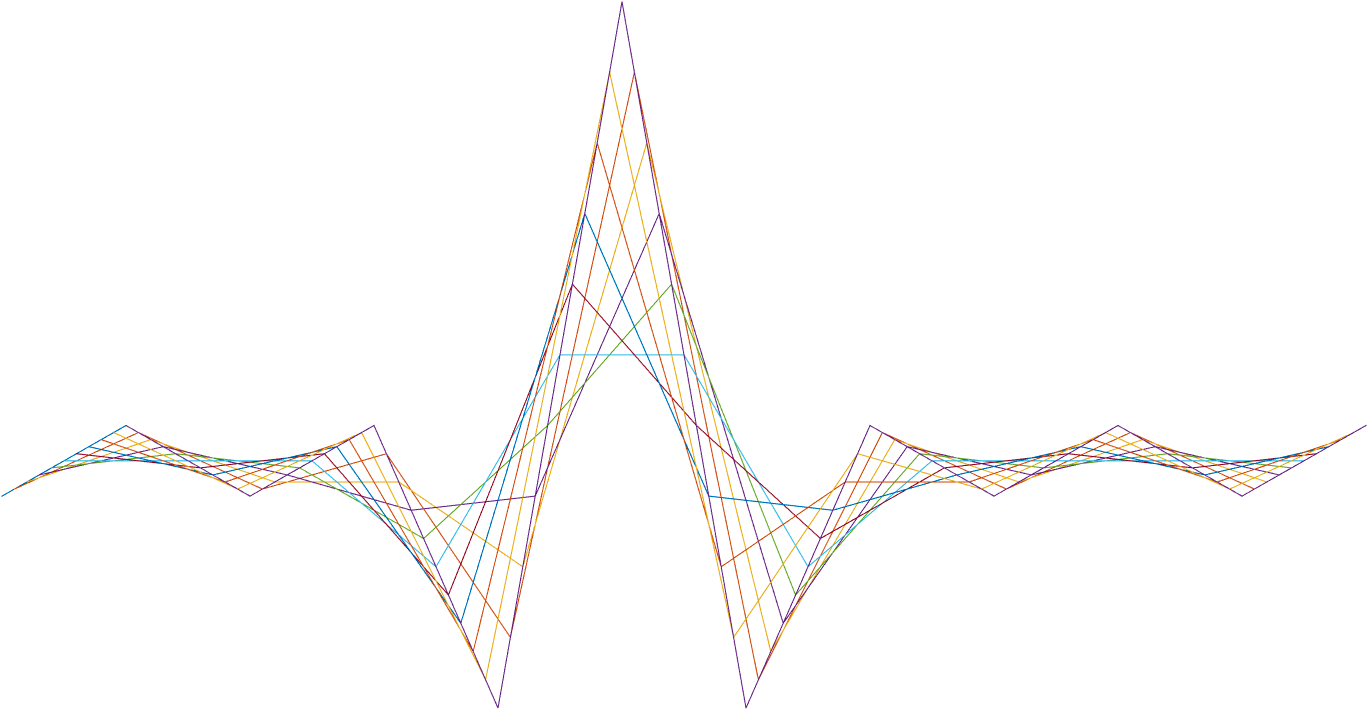
IVCAM2.0 3D Imaging Camera



ASIC A0 JFIL sort filter specification

9 October 2016

Revision 0.5.0

Intel Top Secret

Table 1: Revision history

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Matlab Version | Revision Number | Revised by | Description | Revision Date |
| TBD | 0 | Omer Sella | First draft | 09/06/2016 |
|  |  | Omer Sella | Review by Erez | 13/06/2016 |
|  |  | Omer Sella | Review by Erez and Vitaly | 13/06/2016 |
|  |  | Omer Sella | Polish before presenting + some visio art | 14/06/2016 |
|  |  | Omer Sella | Cleared out the introduction. | 14/06/2016 |
|  |  | Omer Sella | Added a different extension option. | 14/06/2016 |
|  |  | Omer Sella | AI from ASIC review:  Corrected output names.  Unified normalization registers.  Added center pixel invalid handling.  Added bit explanation. | 14/06/2016 |
|  |  | Omer Sella | Revised register names, added “data ==0 implies invalid” handling. | 15/06/2016 |
|  |  | Omer Sella | Added flags | 28/06/2016 |
| 0.6.35 |  | Ohad Menashe | Remove deprecated data paths | 28/8/2016 |
| 0.6.35 |  | Ohad Menashe | Confidence calculation bug: only on valids | 29/8/2016 |
|  |  | Omer Sella | Review | 01/09/16 |
|  |  | Omer Sella | Correction of rejects from review.   * Separated control registers from configuration. * Changed central weight text. * Separated IR from depth. | 12/09/2016 |
| 0.6.37 |  | Omer Sella | Matched doc to code –   * Sort – bypass modes. * Sort – centre weight. | 18/09/2016 |
| 0.7.39 |  | Omer Sella | Added explanation on confidence averaging.  Added a look up table (hard coded) for division when averaging. | 29/09/2016 |
| 0.7.40 |  | Omer Sella | Following a code review with Ohad the following changes were made:   1. Sort changes confidence only of invalid depth pixels. i.e.: if the pixel was valid, but sort was performed, its confidence does not change. 2. Division table: The numbers by which we multiply in order to average the confidence were changed to satisfy that the average when all valid pixels have confidence 15 – remains 15. |  |
| 0.7.41 |  | Omer Sella | Scan hole exception:  Added handling of the case where there are no valid pixels in a 3X3 window. | 09/10/2016 |
| 0.7.41 |  | Omer Sella | * Corrected division LUT values. * Added – need to add 512 to the result of nominator before bitshifting by 10. * Added 0 3X3 window handling. | 10/10/2016 |

Contents

[1 Introduction 1](#_Toc452908940)

[2 Interfaces 1](#_Toc452908941)

[2.1.1 Input 1](#_Toc452908942)

[2.1.2 Output 1](#_Toc452908943)

[3 Detailed description 1](#_Toc452908944)

[3.1 Functionality description 1 1](#_Toc452908945)

[3.1.1 Sub function 1](#_Toc452908946)

[3.1.2 Sub function 1](#_Toc452908947)

[3.2 Functionality description 2 1](#_Toc452908948)

[4 Appendix A: Stuff that goes in the appendix 1](#_Toc452908949)

[5 Registers 1](#_Toc452908950)

[6 Memory 1](#_Toc452908951)

[7 Performance 1](#_Toc452908952)

[8 Test plan 1](#_Toc452908953)

Tables

[Table 1: Revision history 1](#_Toc461380417)

[Table 2: Registers 7](#_Toc461380418)

Introduction

Sort filter - the purpose of this block is to have a weighted average of the input pixels. While the weights are written in register space in a fixed order, the multiplication of a pixel by a weight is only done after the pixels are sorted by ascending values (and where invalid pixels are dealt with differently, see explanation later on).

The filter works on a 3X3 sliding window of depth, IR and confidence.

Denote the weights as the vector - so nine weights indexed from -4 to 4 both for depth and IR, and the incoming pixels as:

The output we seek is the result of where .

Interfaces

Input

* 3x3 16b depth
* 3x3 12b IR
* 3x3 4b confidence
* 4b flags

Output

* 16b depth
* 12b IR
* 4b confidence
* 4b flags
* Memory

Sort is a 3X3 sliding window, and so it uses 2 columns from the buffer.

Detailed description

This function applies a sort function on both depth and IR separately, while considering the confidence values as a mask.

Pixels with confidence value 0 are invalid pixels.  
As inputs to this block contains both IR and depth, the block has a separate bypass for either valid IR or valid depth

In addition each stream (IR, depth) has two modes of operation, one mode that performs sort on all pixels (valid or not), while the other mode performs sort only on invalid pixels.

So to conclude the modes of operation we have (same as in Edge):

|  |  |  |  |
| --- | --- | --- | --- |
| Register name | Value in binary | Bypass sort | Mode of operation |
| RegsJFIL\*bypassMode | 3’bXX1 | Yes | None. |
| RegsJFIL\*bypassMode | 3’b000 | No | Apply on all pixels. |
| RegsJFIL\*bypassMode | 3’b010 | No | Do not apply on valid IR pixels, apply on all depth pixels. |
| RegsJFIL\*bypassMode | 3’b100 | No | Do not apply on valid depth pixels, apply on all IR pixels. |
| RegsJFIL\*bypassMode | 3’b110 | No | Do not apply on valid IR pixels, do not apply on valid depth pixels. |

Important note: Confidence is only updated in case that depth is acted ON AN INVALID PIXEL.. In other words: confidence is updated if and only if a sort operation on depth is performed ON AN INVALID PIXEL. It may be possible that sort only acts on IR, in which case confidence is untouched.

The updated confidence depends on the register RegsJFILsort\*doConfAveraging

If this register is set to 1, we update the confidence to be the average of the confidence of the valid pixels.

Since averaging requires division by one of the numbers: 1,2,3,4,5,6,7,8,9 we use a multiplication by constant and bit shifting by 10 (equivalent to division by 1024) after adding 512 to the result.

The following table explains how to do the division:

|  |  |  |
| --- | --- | --- |
| Number of valid pixels in the window  V | Multiply the sum of valids by  M | bitshift(V\*15\*M + 512,-10) |
| 0 | 0 | -- |
| 1 | 1024 | 15 |
| 2 | 512 | 15 |
| 3 | 342 | 15 |
| 4 | 256 | 15 |
| 5 | 205 | 15 |
| 6 | 171 | 15 |
| 7 | 146 | 15 |
| 8 | 128 | 15 |
| 9 | 114 | 15 |

Otherwise, we set the new confidence to be the value listed in RegsJFILsort\*FixedConfValue

We mark the weights as:

where:

W= JFILsort\*dWeightsSide

With byte ordering

{

So for example 0x0040 is interpreted as {.

C= JFILsort\*dWeightsCentre

When operating on depth, and where:

W= JFILsort\*iWeightsSide

With byte ordering

{

So for example 0x0040 is interpreted as {.

C= JFILsort\*iWeightsCentre

When operating on IR.

Weights

The weights are given as 9 weights values ranging from 0 to 256 corresponding to . We apply with being the centre weight. The limitations on the weight vectors:



Using these constraints, we can define the weights vector using 4 values defined in the 32 bit register RegsJFILsort\*dWeightsSide for the depth case and RegsJFILsort\*iWeightsSide for the IR case, as well as RegsJFILsort\*dWeightsCentre and RegsJFILsort\*iWeightsCentre for the corresponding centre weight .

The registers RegsJFILsort\*iWeightsCentre and RegsJFILsort\*dWeightsCentre are calculated by FW, by subtracting , i.e.: the centre weight is always the completion to 256.

We take to be the first two nibbles of the 32 register, to be the second two nibbles etc.

Center weight duplication

As previously stated, the sort centers the valid pixels such that invalid pixels are on the edges of the sorted vector. In order to make use of all the weights, weights that correspond to invalid pixels are summed and added to . This way the sum of the used weights remains 256.

Zero depth result

It may be possible that a sort operation on a 3X3 window will result in a 0 depth pixel.

Examples may include:

1. Non-zero weights were aligned with zero depth.
2. **All pixels had 0 depth but the register** RegsJFILsort\*doConfAveraging **was equal to 0.**

This is why an additional check of the resulting depth needs to be done, and in case the resulting depth is 0, set the confidence to 0.

Registers

Table 2: Registers

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Name** | **Size** | **Default** | **Range** | **Special values/ description** |
| GNRL |  |  |  |  |
| JFIL |  |  |  |  |
| RegsJFILsort\*bypassMode | 4 | 110(binary) | Valid values:  001  000  110  100  010 | 001 Bypass  000 Apply on all.  110 do not apply on valid IR/depth  100 – do not apply on valid depth  010 – do not apply on valid IR  Other modes are invalid. |
| RegsJFILsort\*iWweightsSide | 32 | 000040  (hex) | 4x[00-ff] |  |
| RegsJFILsort\*iWeightsCentre |  |  |  | Auto Generated |
| RegsJFILsort\*dWeightsSid | 32 | 000040  (hex) | 4x[00-ff] |  |
| RegsJFILsort\*dWeightsCentre |  |  |  | Auto Generated |
| RegsJFILsort\*FixedConfValue | 4 | 0 | [0-15] |  |
| RegsJFILsort\*doConfAveraging | 1 | 1 | 0/1 |  |

Test plan

|  |  |  |
| --- | --- | --- |
| **Name** | **Value** | **Distribution** |
| GNRL |  |  |
| JFIL |  |  |
| RegsJFIL\*bypassMode | 001  000  110  100  010 | 4%  24%  24%  24%  24% |
| RegsJFILsort\*iWweightsSide | This is a vector of 4 values. each should be checked from 0 to 255 under the constraint that their sum is between 0 and 256 | 100% |
| RegsJFILsort\*iWeightsCentre | FW generated |  |
| RegsJFILsort\*dWeightsSide | This is a vector of 4 values. each should be checked from 0 to 255 under the constraint that their sum is between 0 and 256 | 100% |
| RegsJFILsort\*dWeightsCentre | FW generated |  |
| RegsJFILsort\*FixedConfValue | 0 | 1% |
| 1 | 20% |
| 2 | 20% |
| 3 | 20% |
| 4-15 | 49% |
| RegsJFILsort\*doConfAveraging | 0 | 20% |
| 1 | 80% |