Pipelined SiFH

Jia Shi

Faculty of Electrical Engineering, Mathematics and Computer Science
Delft University of Technology
E-mail: J.Shi-6@student.tudelft.nl
Student Number: 5218845

1 Shifted inter-frame histograms (SiFH)

Two histograms are saved with two detections in shifted inter-frame histograms (SiFH). The first one is called coarse histogram (CH), with a bin number of 2^{N_b}. CH will be built with the result of the first detection. The second one, called fine histogram (FH), will be made with the result of the second detection. CH finds the peak position roughly, then FH zoom in the peak position of the first histogram and finds a more accurate peak result. The number of bits of the CH and FH is both 2^{N_b}, but FH has a smaller gap between each bin to realize an accuracy with 2^{N_p}. Because of the same bin number, CH and FH can share the same part of the memory. The detail about how CH and FH has been built could be seen in Fig.**??**

Before mapping the second Nb-bit histogram (FH), the incoming pixels have to pass through the filter to extract the input data near the peak.

$$TH_- \ \geq pixel\ value > TH_+ \tag{1}$$

The two thresholds is depend on the result of CH and N_b, N_p. Assume the peak of CH is P_{CH}:

$$TH_- = 2^{N_p - N_b} P_{CH} \ + \ SB \ ; \ TH_+ = 2^{N_p - N_b} P_{CH} \ - \ SB \tag{2}$$

SB is the number of side-band bins.

$$SB \ = \ 2^{N_b - 1} \tag{3}$$

To map the 2^{N_p} bits inputs into a 2^{N_b} bit histogram, a shifting value Δ is used to shift the inputs:

$$\Delta = TH_- = 2^{N_p - N_b} P_{CH} \ + \ SB \tag{4}$$

After the peak position of FH (P_{FH}) is found, it will shifted back (shifted upwards) to compute the real result:

$$real\ result = P_{FH} + \Delta \tag{5}$$

2 Hardware implementation

2.1 Variables

Variables in this design:

- ***Nb*** : the bit precision of first histogram.
- ***Np*** : the bit precision of second histogram.
- ***INPUT_MAX*** : the max value of input rough data. ***INPUT_MAX*** = $\sum_{n=1}^{Np} 2^n$
- ***PIXEL_NUM*** : Overall pixel number in sensor.
- ***SRAM_NUM***: the number of pixels that work in parallel.
- ***PIXEL_NUM_PER_RAM*** : number of pixels that save in one SRAM. ***PIXEL_NUM_PER_RAM*** $=$ ***PIXEL_NUM***$/$***SRAM_NUM***
- ***BIN_NUM_PER_HIS*** : number of bins per histogram. ***BIN_NUM_PER_HIS*** $= 2^{\boldsymbol{Np}}$
- ***peakMax*** : number of bits per bin.
- ***ACQ_NUM*** : number of acquisitions.

The actual parameter for this design is listed in Tab. 3

Table 1: Actual parameter for pipelined SiFH design

parameter	***Nb***	***Np***	***PIXEL_NUM***	***SRAM_NUM***
Value	8	10	200*160	200

parameter	***PIXEL_NUM_PER_RAM***	***BIN_NUM_PER_HIS***	***peakMax***	***ACQ_NUM***
Value	160	64	21	33333

2.2 clk frequency

Using the parameter in 3, assume a 30frame/s frame rate, the time for one frame is 33ms. Since each pixel has two results in one acquisition, a 33333 acquisition number means 33ms/(33333*2), about 0.5μs time for one pixel. ***PIXEL_NUM_PER_RAM*** pixels in serial means 0.5 μs / 160, about 3ns one clk. Hence the chosen frequency is 300M.

2.3 Implementation with pipeline methodology

2.3.1 Pocessing detail of pipeline methodology

Fig. 1 shows the block diagram of pipelined methodology implementation. The overall design follows the timeline, the first and second histograms are built based on the first and second measurements separately. To better explain the processing details, an example is given using the following test-used only data in table 2. The minimum parameters are chosen to present the detail of processing. Though only one pixel is been processed, it can present the process clearly since all the other pixels will be processed in the exact same way if there are more pixels.

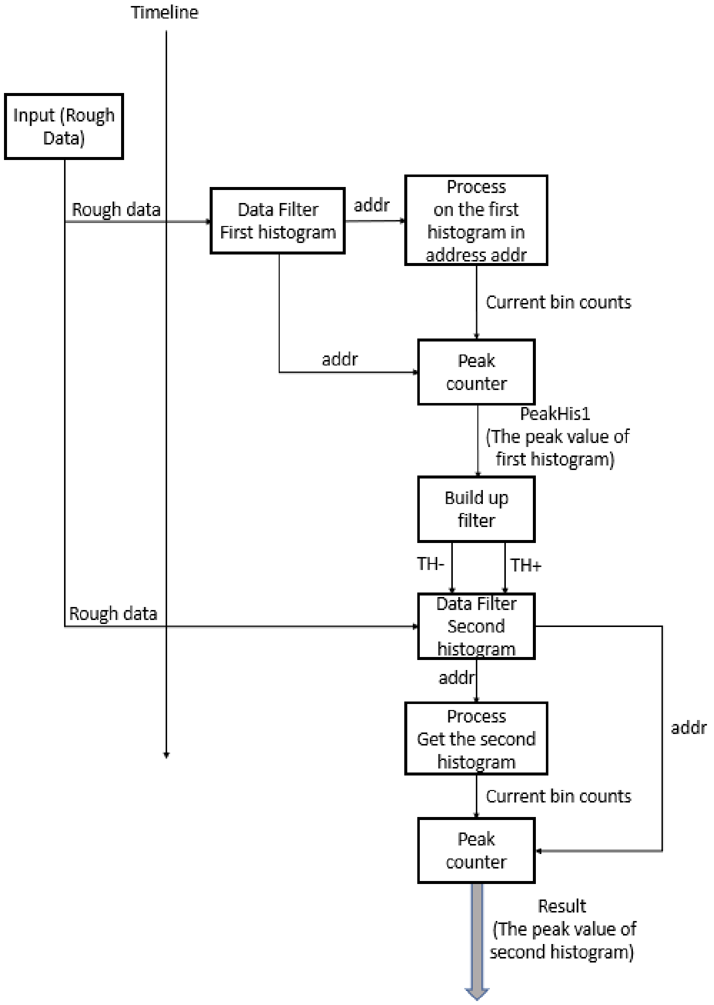

Timeline
Input (Rough Data)
Rough data
Data Filter First histogram
addr
Process on the first histogram in address addr
Current bin counts
addr
Peak counter
PeakHis1 (The peak value of first histogram)
Build up filter
TH-
TH+
Rough data
Data Filter Second histogram
addr
Process Get the second histogram
addr
Current bin counts
Peak counter
Result (The peak value of second histogram)

Figure 1: Block Diagram of pipelined methodology implementation: The input rough data of two measurements will be filtered and built up as two histograms separately. The peak counter finds out the max bin counts in the histogram and outputs the value when the input of a single measurement are all already being processed. The second histogram's filter is being built based on the peak value of the first histogram. 'TH-' is the lower bound of the second histogram's filter, calculated using the peak value of the first histogram. Input smaller than this value will be ignored. 'TH+' is the upper bound of the second histogram's filter, calculated using the peak value of the first histogram. An input larger than this value will be ignored.

Table 2: Example parameter for pipelined SiFH design

parameter	***Nb***	***Np***	***PIXEL_NUM***	***SRAM_NUM***
Value	3	4	1	1

parameter	***PIXEL_NUM_PER_RAM***	***BIN_NUM_PER_HIS***	***peakMax***	***ACQ_NUM***
Value	1	8	3	2

The input rough data range from 0 to ***INPUT_MAX*** - 1. The upper bound of this range presents that this data is not detected successfully and should be neglected.

The data filter of the first histogram shift the rough data and keep the first Nb bits as 'addr'. After filtering, the output 'addr' is corresponding to the Address in SRAM1. SRAM1 is a dual-port SRAM that saves the first histogram after the first measurement, then covered the second histogram during the second measurement. The detail of initial SRAM1 is shown in Fig. 6(a). Each bin in the histogram is presented as an address in SRAM1.

Most SRAM modules do not have a reset signal. There are two different clearing approaches. One is straightforward: clear one address in one clk.SRAM1 has ***PIXEL_NUM_PER_RAM*** * ***BIN_NUM_PER_HIS*** addresses, hence this clearing mechanism introduces a latency ***PIXEL_NUM_PER_RAM*** * ***BIN_NUM_PER_HIS*** clks.

Another method is the signaled clearing mechanism, which does not involve latency because the histogram bins are initialized on the fly when needed. In this method, a special state status register is needed. The status of each bin in SRAM1 is presented by one bit in a status register, saved in stateRAM part in SRAM2. A simple example of stateRAM is shown in Fig.2(c). Suppose that the current 'addr' appears for the first time during the accumulation of a histogram, the 1 bit state bit for this address will be 0. In this case, the data in this address will be written as 1. Otherwise, the data in this address will be read, add 1, then write back. The drawback of this method is that a ***PIXEL_NUM_PER_RAM*** * ***BIN_NUM_PER_HIS*** bits register is needed, which is 200*160 bits typically, cost too much area. To solve this large register problem, [1] introduced a latch-based reset block to save the area of the state status register. However, a latch is unstable and difficult to test, and it is hard to produce this idea as a product. Since my project targets a design that could be mass-produced, this latching method is not implemented. Implementation with pipeline methodology is intended to get as small as possible latency, so the second on-fly refresh method is used.

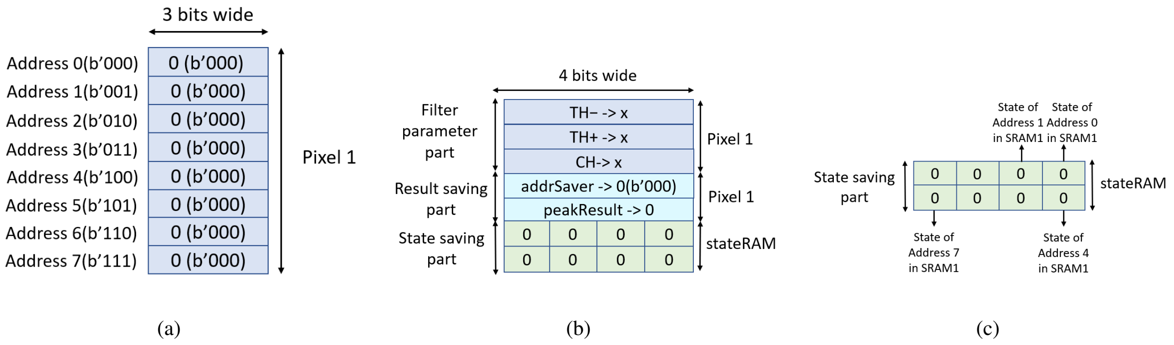
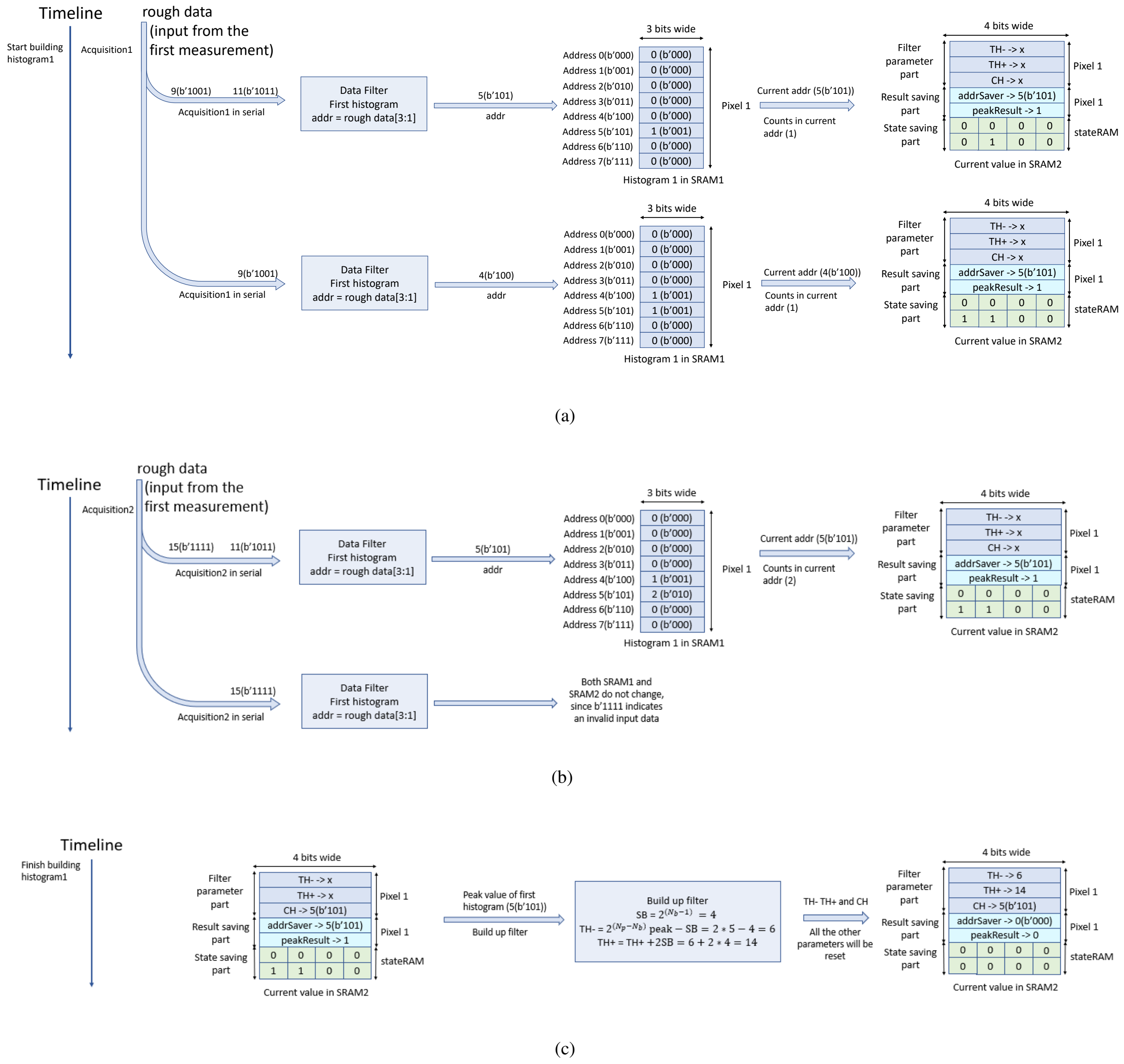


Figure 2: Presentation about the initial data in SRAM1 and SRAM2: (a) shows the initial value in SRAM1. Each Address in SRAM1 counts the number of time that this address is been read. (b) shows the initial value in SRAM2. 'x': means high impedance, no data is saved. CH is the peak value of first histogram. The result saving part save the intermediate result of find out the max value that saved in SRAM1. 'addrSaver' is the address of current max value in SRAM1. 'peakResult' is the current max value in SRAM1. (c) shows the detail of state saving part, each bit in state saving part indicates the status of one address in SRAM1.

Each address in SRAM1 counts the number of times that this address is been read. In each cycle, 'peak counter' read the number(old count) in Address 'addr', then write back a new count after adding 1 to the older one. If the new count is larger than SRAM2, this count and the current 'addr' will be saved into the Result saving part in SRAM2, which is a part 'peak counter'. Using the example in Fig.3(a), there are two input rough data(11 and 9) in a single acquisition. Since both the two inputs are counted only once, the Result saving part does not change after the second data(9) arrives. In Fig.3(b), the second input(15) is the upper bound of input rough data, meaning this data is not detected, so it is neglected.

When all the data from the first measurement is being processed, 'peak counter' will output a peak value of the first histogram by copying the data in 'addrSaver' into CH. Afterward, the filter of the second histogram will be built as shown in Fig.3(c). The equation that used to calculate filter is equation 2 and equation 3.The calculated filter parameter will be saved in the Filter parameter part in SRAM2.

All the data in SRAM2 will be reset before the process of the second histogram. The process of the second histogram is the same as the process of the first histogram except the input filter is different, the whole process of the second histogram is shown in Fig.4. The filter of the second histogram presents a histogram with $2^N p$ using only $2^N b$ as shown in Fig.5.

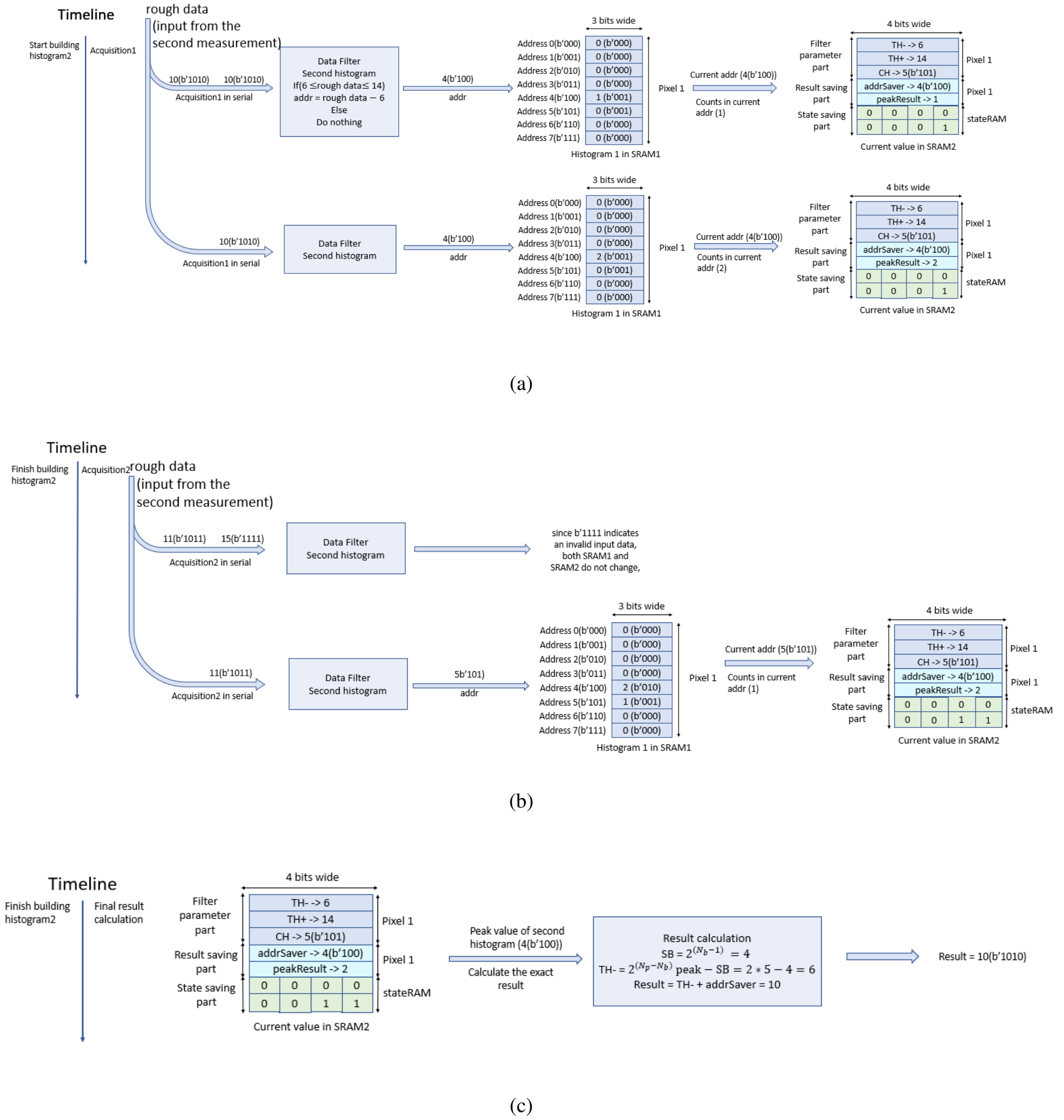


(a)

(b)

(c)

Figure 3: Presentation about the data process of the first measurement: (a) shows the detail when the data of the first acquisition is processed. Each time a new address in SRAM1 has been read, the number of times that address has been read (counts in current addr) will be sent to SRAM2 and written back to the current address in SRAM1. Only when the current counts are larger than the counts saved in the 'peak result' part, the 'addrSaver' and 'peak result' in SRAM2 will be refreshed with the current result. Otherwise, they will remain the same. (b) shows the detail when the data of the second acquisition is processed. SRAM1 is not refreshed and keeps counting with counts in the first acquisition. (c) shows the building of the second histogram's filter. All the other parameters saved in SRAM2 except 'TH-,' 'TH+' and 'CH' and all refreshed.



(a)

(b)

(c)

Figure 4: Presentation about the data processing of the second measurement: The (a) shows the detail when the data of the first acquisition is processed. The processing detail is the same as the way data of the first measurement is processed expect a different data filter is used. (b) shows the detail when the data of the second acquisition is processed. (b) shows the final result calculation when the process for all data is done.

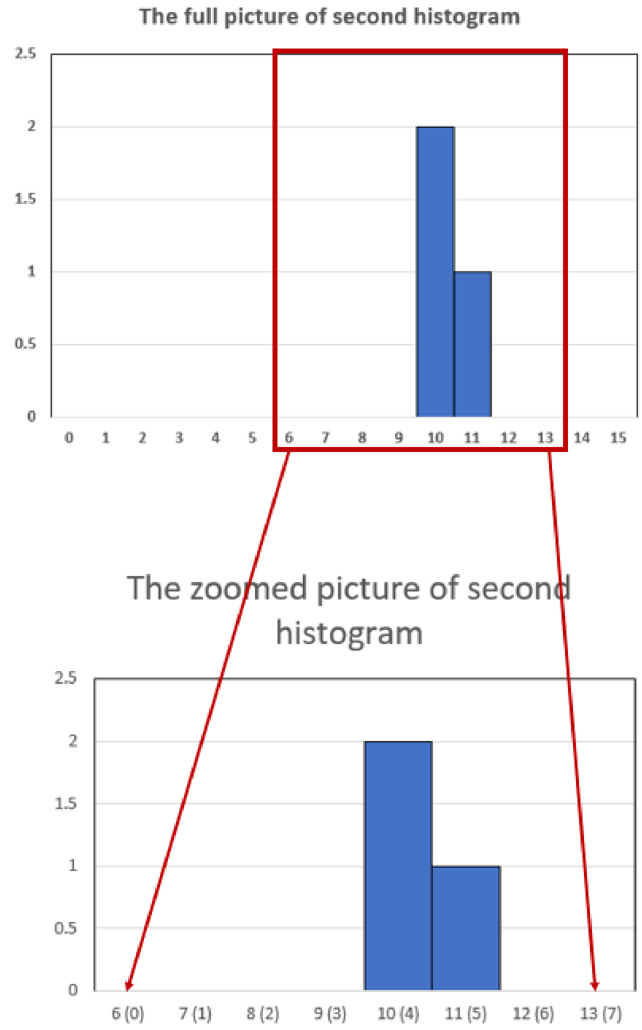
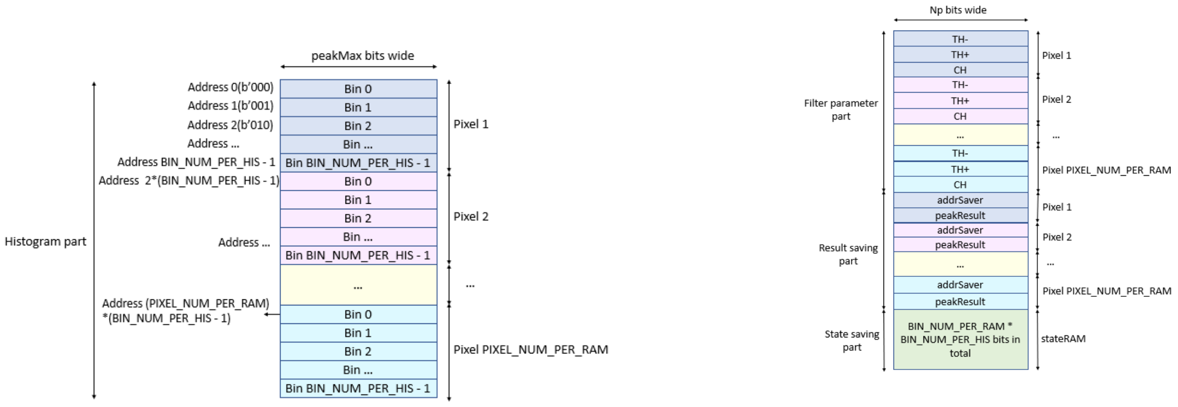


Figure 5: The full-picture of the second histogram: The upper histogram is the complete picture of the second histogram. The input filter cut a part of the second histogram out by saving the bins from 6 to 13 into addresses 0 to 7 in SRAM1. The result of the first histogram defines the upper and lower bound of zoomed-in filter

2.3.2 SRAM cost calculation

A pair of SRAM1 and SRAM2 with unknown parameters is shown in 6, the way data is processed precisely the same as the one in the former example.



(a)

(b)

Figure 6: The structure of data in one pair of SRAM: (a) shows the memory location in SRAM1. In the histogram part of SRAM1, each address (Bins) counts the number of times that this address (Bins) has been read. (b)shows the memory location of SRAM2. The filter part saves the parameters to build up the filter of the first histogram.

A brief calculation of SRAM size is shown below. All the size memory blocks (SRAM and registers) are in bits during the calculation. S_{his}, Assume the size of SRAM1 is S1 bits, the size of SRAM2 is S2 bits. They could be calculated with the following equation:

$$S_{his} = \textbf{\textit{peakMax}} * \textbf{\textit{BIN\_NUM\_PER\_HIS}} * \textbf{\textit{SRAM\_NUM}} \tag{6}$$

$$S_{result} = 2 * \textbf{\textit{Np}} * \textbf{\textit{PIXEL\_NUM\_PER\_RAM}} \tag{7}$$

$$S_1 = S_{his} * \textbf{\textit{SRAM\_NUM}} \tag{8}$$

$$S_{result} = \textbf{\textit{Np}} * 2 * \textbf{\textit{PIXEL\_NUM\_PER\_RAM}} \tag{9}$$

$$S_{filter} = \textbf{\textit{Np}} * 3 * \textbf{\textit{PIXEL\_NUM\_PER\_RAM}} \tag{10}$$

$$S_2 = (S_{result} + S_{filter}) * \textbf{\textit{SRAM\_NUM}} \tag{11}$$

Most SRAM modules do not have a reset signal. There are two different clearing approaches. One is straightforward: clear one address in one clk. SRAM1 and SRAM2 in Fig.**??** can clear their address simotaniously. The filter parameter part do not need to be cleared at the beginning, the data will refresh itself when the calculation is done. Since SRAM1 has ***PIXEL_NUM_PER_RAM*** * ***BIN_NUM_PER_HIS*** addresses, the result saving part in SRAM2 has 3 * ***PIXEL_NUM_PER_RAM*** addresses, this clearing mechanism introduces a latency ***PIXEL_NUM_PER_RAM*** * ***BIN_NUM_PER_HIS*** clks.

Another method is the signaled clearing mechanism, which does not involve latency because the histogram bins are initialized on the fly when needed. In this method, a special state status register is needed. The status of each bin in SRAM1 is presented by one bit in a status register. Suppose that the current ADDR appears for the first time during the accumulation of a histogram, the 1 bit state bit for this address will be 0. In this case, the data in this address will be written as 1. Otherwise, the data in this address will be read, add 1, then write back. However, in this method, a ***PIXEL_NUM_PER_RAM*** * ***BIN_NUM_PER_HIS*** bits register is needed, which is 200*160 bits typically, cost too much area. To solve this large register problem, [1] introduced a latch-based reset block to save the area of the state status register. However, a latch is unstable and difficult to test, and it is hard to produce this idea as a product. Since my project targets a design that could be mass-produced, this latching method is not implemented.

It is possible to save the state status in SRAM2, using ***PIXEL_NUM_PER_RAM*** * ***BIN_NUM_PER_HIS / Np*** addresses. The size of state saving part in SRAM2 S_{state} is at least:

$$S_{state} = \textbf{\textit{BIN\_NUM\_PER\_HIS}} * \textbf{\textit{PIXEL\_NUM\_PER\_RAM}} \tag{12}$$

The actual size of the state saving part will be slightly larger than S_{state} since the width of SRAM is fixed as Np. Then, since the histogram part in SRAM1 no longer need to be refreshed, the clearing mechanism introduces a latency 3 * ***PIXEL_NUM_PER_RAM + PIXEL_NUM_PER_RAM*** * ***BIN_NUM_PER_HIS / Np*** clks.

The second cleaning methodology is used for the pipelined implementation. The calculation delay (the period between the end of an input signal and the ready of output result) is 3 clk.

$$L_{pipelined} = 3 * \textbf{\textit{PIXEL\_NUM\_PER\_RAM}} + \textbf{\textit{PIXEL\_NUM\_PER\_RAM}} * \textbf{\textit{BIN\_NUM\_PER\_HIS}}/\textbf{\textit{Np}} \tag{13}$$

Using the parameter in Tab. 3, the time that used to reset SRAM modules $L_{pipelined}$ is :

$$L_{pipelined} = 3 * 160 + \frac{(64 * 160)}{10} = 480 + 1024 = 1504cycles$$

For the area, using the parameter shown in table 3, we can get the following result:

$$S_{his} = \textbf{\textit{peakMax}} * \textbf{\textit{BIN\_NUM\_PER\_HIS}} * \textbf{\textit{SRAM\_NUM}} = 26.25KB$$

$$S_{result} = 2 * \textbf{\textit{Np}} * \textbf{\textit{PIXEL\_NUM\_PER\_RAM}} = 400B$$

$$S_{filter} = 3 * \textbf{\textit{Nb}} * \textbf{\textit{BIN\_NUM\_PER\_HIS}} * \textbf{\textit{PIXEL\_NUM\_PER\_RAM}} = 600B$$

$$S_1 = S_{his} * \textbf{\textit{SRAM\_NUM}} = 5550KB \approx 5.13MB$$

$$S_2 = \textbf{\textit{Np}} * 5 * \textbf{\textit{PIXEL\_NUM\_PER\_RAM}} * \textbf{\textit{SRAM\_NUM}} = 195.31KB$$

$$S_{state} = \textbf{\textit{BIN\_NUM\_PER\_HIS}} * \textbf{\textit{PIXEL\_NUM\_PER\_RAM}} = 1.25KB$$

$$S_{all} = S_1 + S_2 + S_{state} = 5445.31KB \approx 5.32MB$$

2.4 Implementation without pipeline methodology

2.4.1 Pocessing detail of methodology without pipeline

The most apparent difference between implementation with and without pipeline methodology is that the peak Detector part without pipeline implementation only starts to calculate when all the bins in histograms stop counting. Hence the implementation without pipeline methodology face a huge delay that a pipelined implementation will not face: read the bin counts in each address from SRAM1 in Fig.6(a). This counts reading process will cost ***PIXEL_NUM_PER_RAM*** * ***BIN_NUM_PER_HIS*** clks.

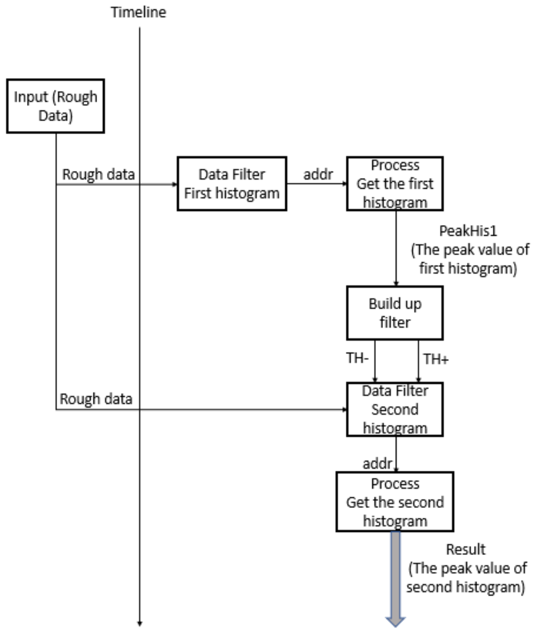
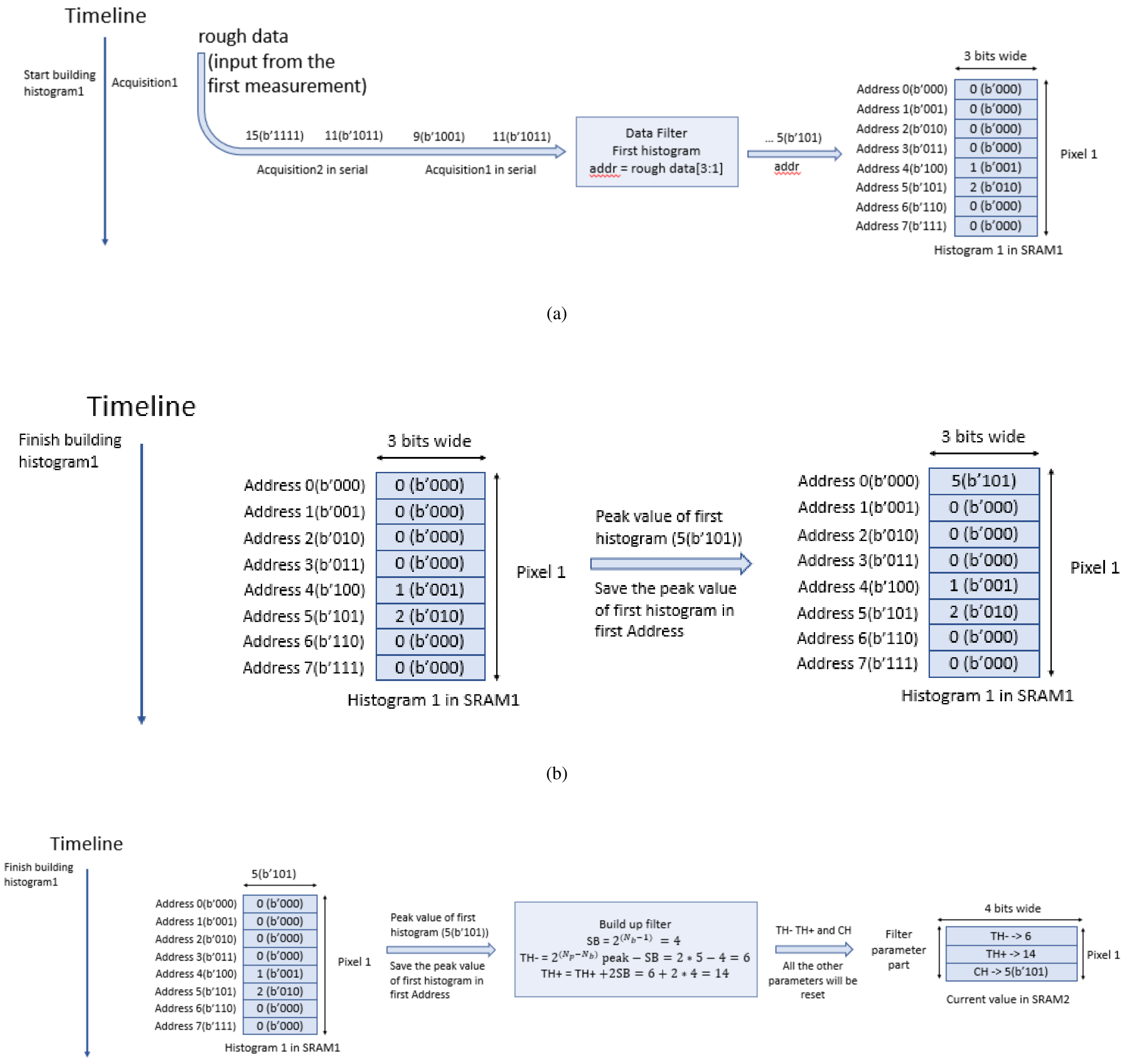

Timeline
Input (Rough Data)
Rough data
Data Filter First histogram
addr
Process Get the first histogram
PeakHis1 (The peak value of first histogram)
Build up filter
TH-
TH+
Rough data
Data Filter Second histogram
addr
Process Get the second histogram
Result (The peak value of second histogram)

Figure 7: The block diagram of implementation without pipelined methodology: The peak value will be find until all the input of one single measurement is being processed.

An example with the same input data as the example in section 2.3 in the version of implementation without pipeline is shown in Fig.8. Instead of refreshing the current peak value of every signal, the result will be saved in the first address of SRAM1 after the histogram of all pixels is built. Both the result saving part and state saving part in SRAM2 in Fig.2(b) are not used anymore.

The process of the second histogram is shown in Fig.9. The whole SRAM1 is cleared before building the second histogram.

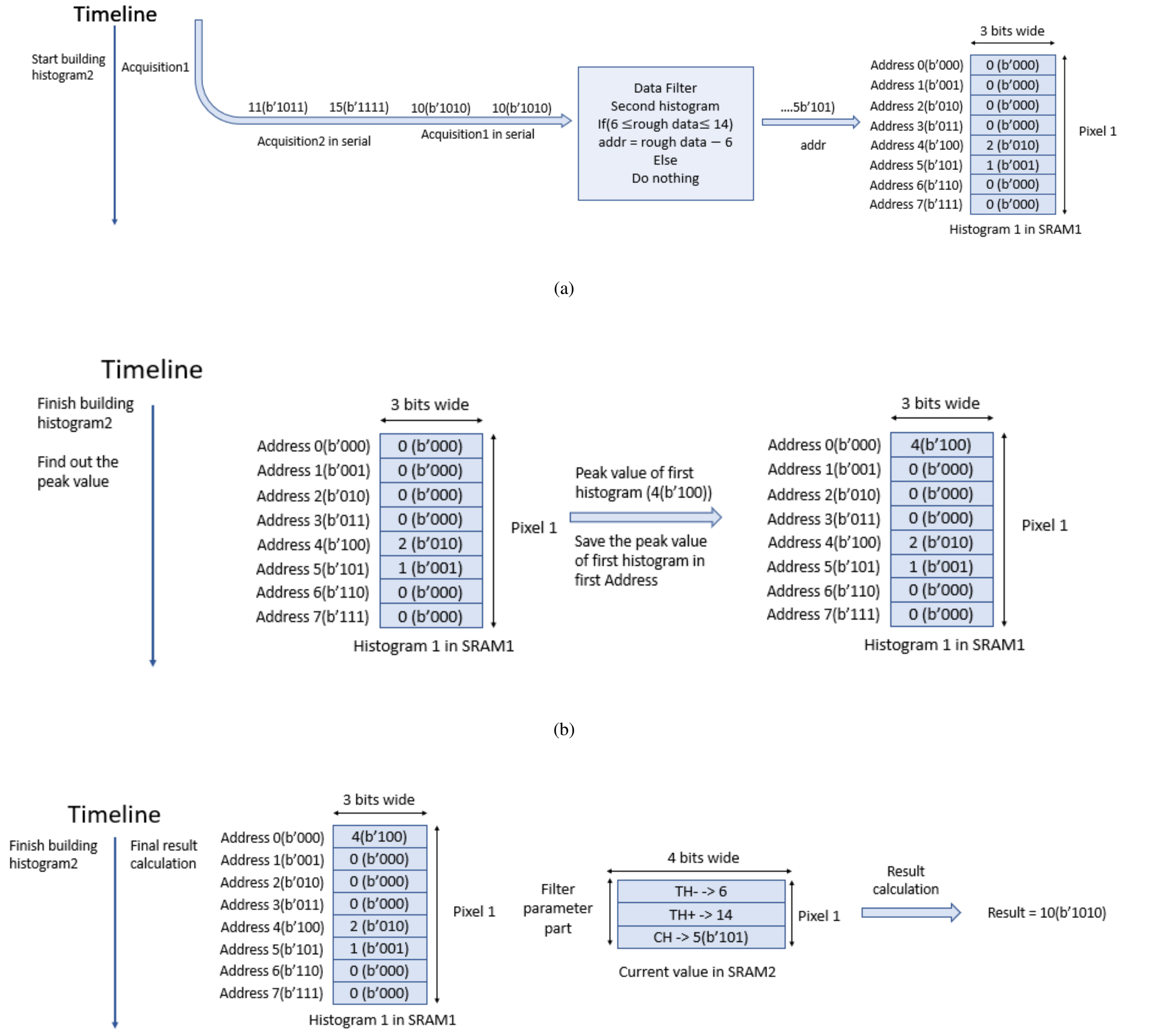

Timeline
Start building histogram1
Acquisition1
rough data (input from the first measurement)
15(b'1111) 11(b'1011) 9(b'1001) 11(b'1011)
Acquisition2 in serial
Acquisition1 in serial
Data Filter
First histogram
addr = rough data[3:1]
... 5(b'101)
addr
3 bits wide
Address 0(b'000) 0 (b'000)
Address 1(b'001) 0 (b'000)
Address 2(b'010) 0 (b'000)
Address 3(b'011) 0 (b'000)
Address 4(b'100) 1 (b'001)
Address 5(b'101) 2 (b'010)
Address 6(b'110) 0 (b'000)
Address 7(b'111) 0 (b'000)
Pixel 1
Histogram 1 in SRAM1
Timeline
Finish building histogram1
3 bits wide
Address 0(b'000) 0 (b'000)
Address 1(b'001) 0 (b'000)
Address 2(b'010) 0 (b'000)
Address 3(b'011) 0 (b'000)
Address 4(b'100) 1 (b'001)
Address 5(b'101) 2 (b'010)
Address 6(b'110) 0 (b'000)
Address 7(b'111) 0 (b'000)
Pixel 1
Histogram 1 in SRAM1
Peak value of first histogram (5(b'101))
Save the peak value of first histogram in first Address
3 bits wide
Address 0(b'000) 5(b'101)
Address 1(b'001) 0 (b'000)
Address 2(b'010) 0 (b'000)
Address 3(b'011) 0 (b'000)
Address 4(b'100) 1 (b'001)
Address 5(b'101) 2 (b'010)
Address 6(b'110) 0 (b'000)
Address 7(b'111) 0 (b'000)
Pixel 1
Histogram 1 in SRAM1
Timeline
Finish building histogram1
5(b'101)
Address 0(b'000) 0 (b'000)
Address 1(b'001) 0 (b'000)
Address 2(b'010) 0 (b'000)
Address 3(b'011) 0 (b'000)
Address 4(b'100) 1 (b'001)
Address 5(b'101) 2 (b'010)
Address 6(b'110) 0 (b'000)
Address 7(b'111) 0 (b'000)
Pixel 1
Histogram 1 in SRAM1
Peak value of first histogram (5(b'101))
Save the peak value of first histogram in first Address
Build up filter
SB = $2^{(N_b-1)}$ = 4
TH- = $2^{(N_p-N_b)}$ peak − SB = 2 ∗ 5 − 4 = 6
TH+ = TH+ +2SB = 6 + 2 ∗ 4 = 14
TH- TH+ and CH
All the other parameters will be reset
Filter parameter part
4 bits wide
TH- -> 6
TH+ -> 14
CH -> 5(b'101)
Pixel 1
Current value in SRAM2

(a)

(b)

(c)

Figure 8: Presentation about the data process of the first measurement in implementation without pipelined methodology: (a) shows the detail when the data of the first measurement is processed. SRAM1 is refreshed before all the operations. No other operation occurs before all the input data in the first measurement are counted in SRAM1 (b) shows the detail when the address of the max value saved in SRAM1 is saved in SRAM1's Address 0. (c) shows the building of the second histogram's filter.


Timeline
Start building histogram2
Acquisition1
11(b’1011)
15(b’1111)
10(b’1010)
10(b’1010)
Acquisition2 in serial
Acquisition1 in serial
Data Filter
Second histogram
If(6 ≤rough data≤ 14)
addr = rough data − 6
Else
Do nothing
....5b’101)
addr
3 bits wide
Address 0(b’000) 0 (b’000)
Address 1(b’001) 0 (b’000)
Address 2(b’010) 0 (b’000)
Address 3(b’011) 0 (b’000)
Address 4(b’100) 2 (b’010)
Address 5(b’101) 1 (b’001)
Address 6(b’110) 0 (b’000)
Address 7(b’111) 0 (b’000)
Pixel 1
Histogram 1 in SRAM1

(a)

Timeline
Finish building histogram2
Find out the peak value
3 bits wide
Address 0(b’000) 0 (b’000)
Address 1(b’001) 0 (b’000)
Address 2(b’010) 0 (b’000)
Address 3(b’011) 0 (b’000)
Address 4(b’100) 2 (b’010)
Address 5(b’101) 1 (b’001)
Address 6(b’110) 0 (b’000)
Address 7(b’111) 0 (b’000)
Pixel 1
Histogram 1 in SRAM1
Peak value of first histogram (4(b’100))
Save the peak value of first histogram in first Address
3 bits wide
Address 0(b’000) 4(b’100)
Address 1(b’001) 0 (b’000)
Address 2(b’010) 0 (b’000)
Address 3(b’011) 0 (b’000)
Address 4(b’100) 2 (b’010)
Address 5(b’101) 1 (b’001)
Address 6(b’110) 0 (b’000)
Address 7(b’111) 0 (b’000)
Pixel 1
Histogram 1 in SRAM1

(b)

Timeline
Finish building histogram2
Final result calculation
3 bits wide
Address 0(b’000) 4(b’100)
Address 1(b’001) 0 (b’000)
Address 2(b’010) 0 (b’000)
Address 3(b’011) 0 (b’000)
Address 4(b’100) 2 (b’010)
Address 5(b’101) 1 (b’001)
Address 6(b’110) 0 (b’000)
Address 7(b’111) 0 (b’000)
Pixel 1
Histogram 1 in SRAM1
Filter parameter part
4 bits wide
TH- -> 6
TH+ -> 14
CH -> 5(b’101)
Pixel 1
Current value in SRAM2
Result calculation
Result = 10(b’1010)

(c)

Figure 9: Presentation about the data process of the second measurement in implementation without pipelined methodology: (a) shows the detail when the data of the second measurement is processed. (b) shows the detail when the address of the max value saved in SRAM1 is saved. (a) and (b) in the second measurement is the same as what that done in the first measurement, except a new filter for input data is used(c) shows the result calculation part, the calculation function is the same as the one used in the pipelined implementation.

In the pipelined implementation, the result saving part will refresh each time if there is any change in the histogram part. However, in the implementation without pipeline methodology, the histogram saving part in SRAM will be refreshed at the beginning.

$$L_{not\_pipelined} = 2 * \textbf{\textit{PIXEL\_NUM\_PER\_RAM}} * \textbf{\textit{BIN\_NUM\_PER\_HIS}} \tag{14}$$

Using the parameter in Tab. 3,

$$L_{not\_pipelined} = 20480 cycles$$

The good thing about not doing pipelining is that the result-saving part in SRAM2 could be saved in the histogram part in SRAM1. Since the result of each pixel is calculated after the whole histogram is built, the result will not change unless the system is being reset. Hence the delete of histogram data will not influence the former calculation. On top of that, the state saving part is not used anymore. The SRAM cost of this implementation could be calculated:

$$S_{not\_pipelined} = S_1 + S_{filter} * \textbf{\textit{SRAM\_NUM}} \tag{15}$$

using the parameter that shown in table 3, we can get following result:

$$S_{not\_pipelined} = S_1 + S_{filter} * \textbf{\textit{SRAM\_NUM}} == 5367.19KB \approx 5.24MB$$

2.5 Expected difference on two implementations

The pipelined implementation reduces the calculation latency by increasing the area's cost.

The saved latency of pipelined implementation could be calculated as follow:

$$L_{saved} = L_{not\_pipelined} - L_{pipelined} = 20480 - 1504 = 8736 cycles \tag{16}$$

If using a 300MHz clk, the L_{saved} is 116.556μs. Though the L_{saved} is a relatively long time, the time that is waiting for input data costs 33333*160*2 = 10666560 clks, making this 8736 clks neglectable.

The saved area cost of implementation without pipelining is shown as follow:

$$S_{save} = S_{pipelined} - S_{not\_pipelined} = 5445.31KB - 5367.19KB = 78.12KB$$

The percentage of SRAM area saving (P_{save}) could be calculated roughly as:

$$P_{save} = \frac{S_{save}}{S_{pipelined}} \approx 1.435\%$$

Though 1.435% is small, 78.12KB still costs many areas.

The actual parameter for this design is listed in Tab. 3

Table 3: Contrast about two implementations: delay means the period between the end of an input signal and the ready of output result plus the time of initialization. The area is the overall area of SRAM, in MB

parameter	delay(cycles)	area (MB)
pipelined	1504	5.32
not pipelined	20480	5.24

References

[1] I. Vornicu, A. Darie, R. Carmona-Galán, and Á. Rodríguez-Vázquez. Tof estimation based on compressed real-time histogram builder for spad image sensors. In *2019 IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 1–4, 2019.