Optical Flow - Final Project

Team 7 朱祐葳 蘇碁傑 吳宜凡

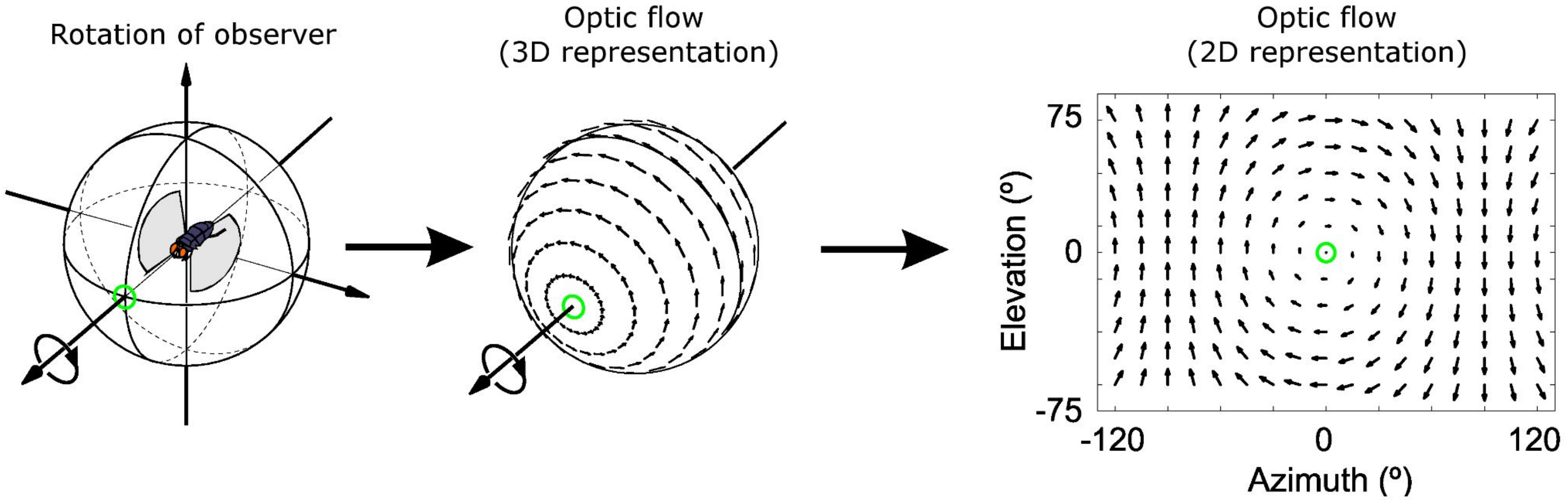
Outline

- Optical Flow introduction
- Kernel Architecture
- Optimization
- Timing & Resource Utilization
- Result

Optical Flow introduction

Optical Flow introduction

Optical flow or **optic flow** is the pattern of **apparent motion** of objects, surfaces, and edges in a visual scene caused by the **relative motion** between an observer and a scene.



Method

- Lucas-Kanade algorithm
  - The Lucas-Kanade optical flow algorithm is a simple technique which can provide an estimate of the movement of interesting features in successive images of a scene.
  - Sparse (some pixels) or dense (all pixels)
  - No longer state-of-the-art but still widely referenced

LK Optical Flow

Unknown motion vector in horizontal/vertical direction

$$I_x v_x + I_y v_y + I_t = 0$$

Spatial derivative

Temporal image brightness derivative

- Weighted least-square fit, assuming constant velocity in each section

$$\sum W^2 [I_x v_x + I_y v_y + I_t]^2 = 0 \quad \Rightarrow \quad \begin{bmatrix} \sum W^2 I_x^2 & \sum W^2 I_x I_y \\ \sum W^2 I_x I_y & \sum W^2 I_y^2 \end{bmatrix} \begin{bmatrix} v_x \\ v_y \end{bmatrix} = - \begin{bmatrix} \sum W^2 I_x I_t \\ \sum W^2 I_y I_t \end{bmatrix}$$

Window function

Constraint

1.The pixel intensity of the object between consecutive frames is constant

2. Similar motion between adjacent pixels

Kernel Architecture

Hardware diagram(8 stage)

Stage1 : Unpack

Stage2 : Gradient xyz

Stage3 : weight_y

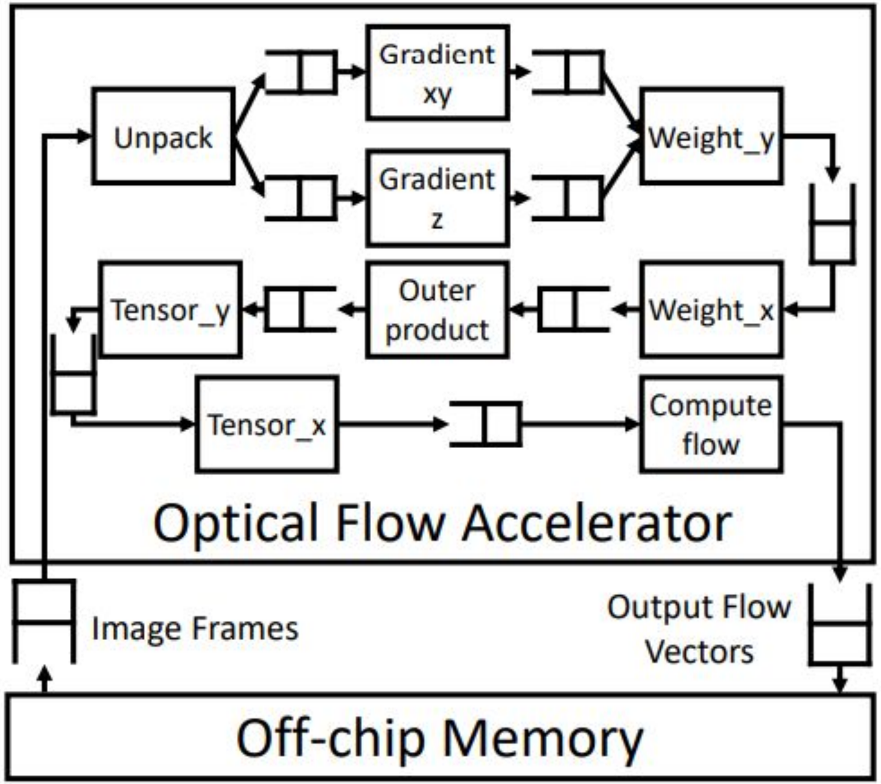
Stage4 : wright_x

Stage5 : outer product

Stage6 : Tensor_y

Stage7 : Tensor_x

Stage8 : Compute flow



Inputs and Outputs

- Input : 5 Consecutive Frame
  - 1024 x 428 Pixel
- Output : Optical flow Result
  - Motion of X and Y
  - FLO file

mp4 ->ppm -> kernel -> flo ->mp4

Optimization

Analysis

- Resource Utilization

Summary

Name	BRAM_18K	DSP48E	FF	LUT	URAM
DSP	-	-	-	-	-
Expression	-	4	0	402	-
FIFO	-	-	-	-	-
Instance	116	60	7848	8121	0
Memory	34816	-	190	0	0
Multiplexer	-	-	-	715	-
Register	-	-	316	-	-
Total	34932	64	8354	9238	0
Available	280	220	106400	53200	0
Utilization (%)	12475	29	7	17	0

- Latency

Summary

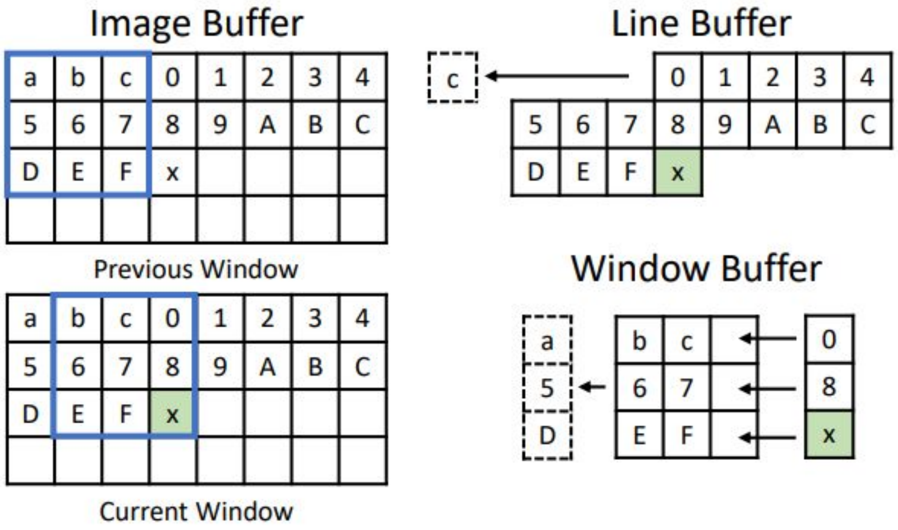
Latency (cycles)		Latency (absolute)		Interval (cycles)		
min	max	min	max	min	max	Type
80958081	295280241	0.810 sec	2.953 sec	80958081	295280241	none

Target

- Memory customization
- Memory bandwidth
- Improve latency

Image line buffer and window buffer

The kernels in optical flow have sliding window access patterns. As a result, line buffer and window buffer are introduced to exploit data reuse and reduce accesses to the next-level memory



Streaming dataflow optimization

After using line buffer and window buffer, the kernels in optical flow have perfectly balanced throughput, and access data in strict sequential order. As a result, execution of all stages can be perfectly overlapped to form a very deep image processing pipeline

Data packing

The hardware function kernel takes in five consecutive image frames. The corresponding pixels in the five frames are packed into a 64-bit integer for fast off-chip data transfer

Pragma

1. pragma HLS array_partition

2. pragma HLS pipeline

3. pragma HLS dependence

4. pragma HLS data_pack

5. pragma HLS dataflow

6. pragma HLS STREAM

Array Partition

- Line Buffer and Window Buffer
- Array Partition
  - Multiple registers instead of one large memory
  - Improves the throughput

```
// calculate gradient in x and y directions
void gradient_xy_calc(input_t frame[MAX_HEIGHT][MAX_WIDTH],
    pixel_t gradient_x[MAX_HEIGHT][MAX_WIDTH],
    pixel_t gradient_y[MAX_HEIGHT][MAX_WIDTH])
{
  // line buffer
  static pixel_t buf[5][MAX_WIDTH];
  #pragma HLS array_partition variable=buf complete dim=1

  // small buffer
  pixel_t smallbuf[5];
  #pragma HLS array_partition variable=smallbuf complete dim=0

  // window buffer
  hls::Window<5,5,input_t> window;
```

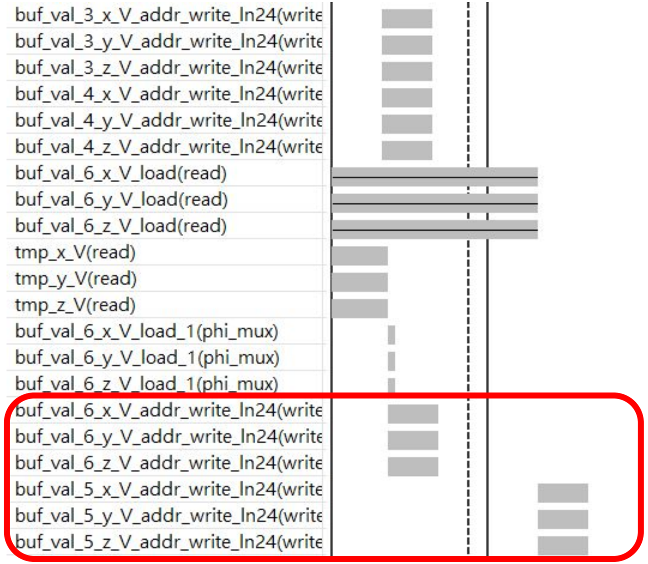
Pipeline

- Read one pixel from input
- Pipeline

```
#pragma HLS pipeline II=1
for (int i = 0; i < 4; i ++ )
    smallbuf[i] = buf[i+1][c];

if (r<MAX_HEIGHT && c<MAX_WIDTH)
  smallbuf[4] = (pixel_t)(frame[r][c]);
else if (c < MAX_WIDTH)
  smallbuf[4] = 0;
```

False Dependence

```
#pragma HLS pipeline II=1
#pragma HLS dependence variable=buf inter false

if(r<MAX_HEIGHT)
{
  buf.shift_pixels_up(c);
  gradient_t tmp;
  tmp.x = gradient_x[r][c];
  tmp.y = gradient_y[r][c];
  tmp.z = gradient_z[r][c];
  buf.insert_bottom_row(tmp,c);
}
else
{
  buf.shift_pixels_up(c);
  gradient_t tmp;
  tmp.x = 0;
  tmp.y = 0;
  tmp.z = 0;
  buf.insert_bottom_row(tmp,c);
}
```

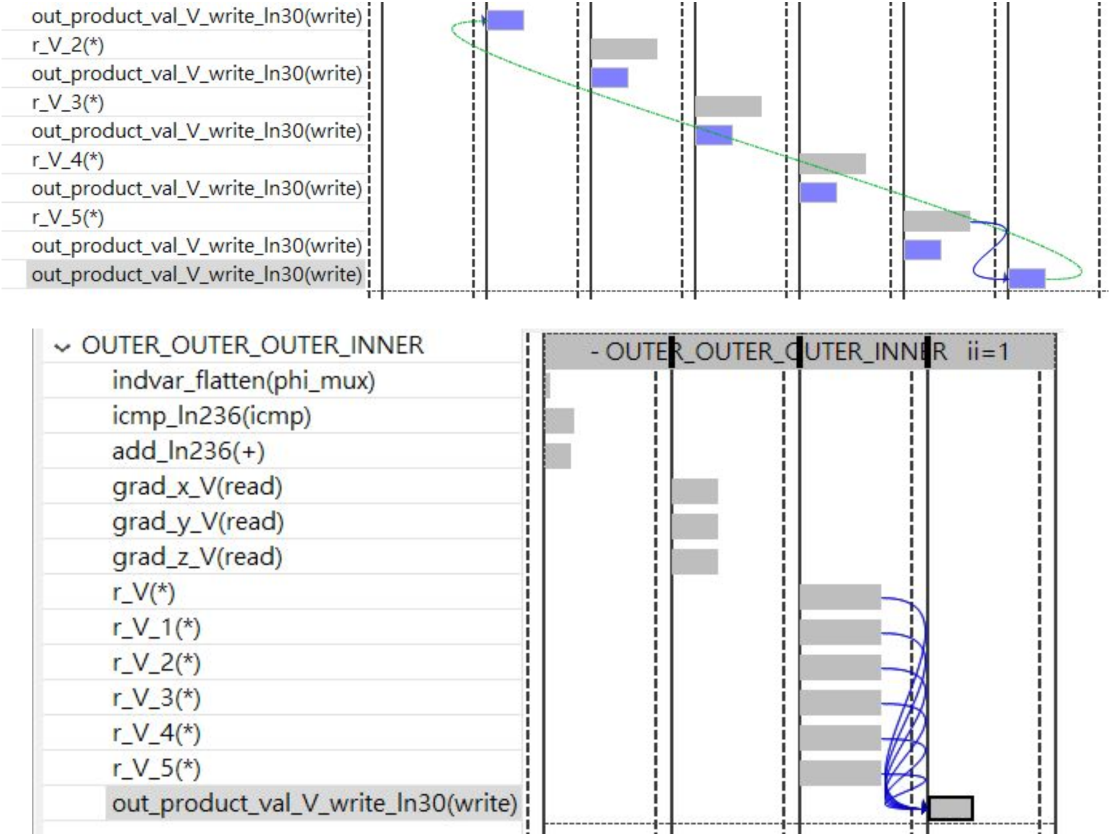


Data Pack

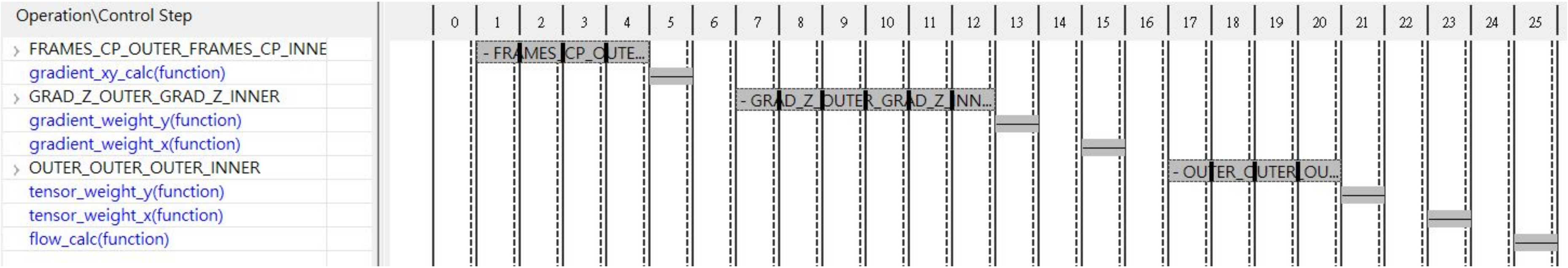
```
#pragma HLS data_pack variable=out_product
#pragma HLS data_pack variable=tensor_y
#pragma HLS data_pack variable=tensor
#pragma HLS data_pack variable=outputs


  gradient_xy_calc(frame3_a, gradient_x, gradient_y);
  gradient_z_calc(frame1_a, frame2_a, frame3_b, frame4_a, frame5_a, gradient_z);
  gradient_weight_y(gradient_x, gradient_y, gradient_z, y_filtered);
  gradient_weight_x(y_filtered, filtered_gradient);
  outer_product(filtered_gradient, out_product);
  tensor_weight_y(out_product, tensor_y);
  tensor_weight_x(tensor_y, tensor);
  flow_calc(tensor, outputs);
```

Pipeline II Violation

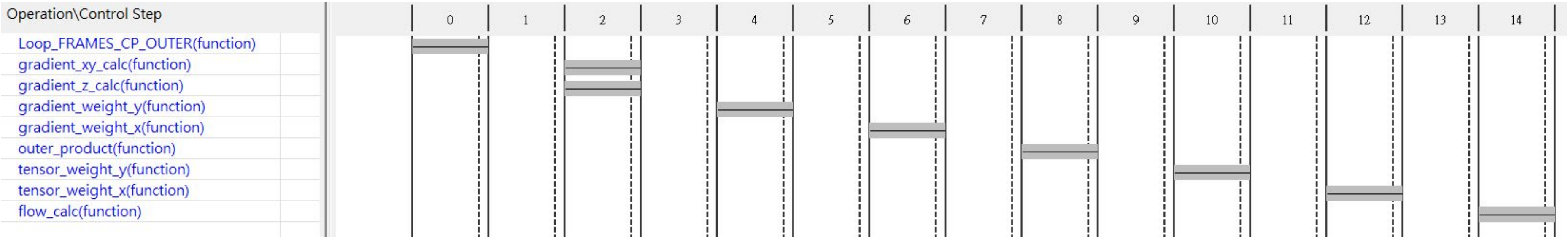
```
#pragma HLS pipeline II=1
gradient_t grad = gradient[r][c];
outer_pixel_t x = (outer_pixel_t) grad.x;
outer_pixel_t y = (outer_pixel_t) grad.y;
outer_pixel_t z = (outer_pixel_t) grad.z;
outer_t out;
out.val[0] = (x*x);
out.val[1] = (y*y);
out.val[2] = (z*z);
out.val[3] = (x*y);
out.val[4] = (x*z);
out.val[5] = (y*z);
outer_product[r][c] = out;
```



Dataflow



Operation\Control Step
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25
FRAMES_CP_OUTER_FRAMES_CP_INNE
gradient_xy_calc(function)
GRAD_Z_OUTER_GRAD_Z_INNER
gradient_weight_y(function)
gradient_weight_x(function)
OUTER_OUTER_OUTER_INNER
tensor_weight_y(function)
tensor_weight_x(function)
flow_calc(function)
- FRAMES_CP_OUTE...
- GRAD_Z_OUTER_GRAD_Z_INN...
- OUTER_OUTER_OU...



Operation\Control Step
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14
Loop_FRAMES_CP_OUTER(function)
gradient_xy_calc(function)
gradient_z_calc(function)
gradient_weight_y(function)
gradient_weight_x(function)
outer_product(function)
tensor_weight_y(function)
tensor_weight_x(function)
flow_calc(function)

Stream

- FIFO instead of RAM

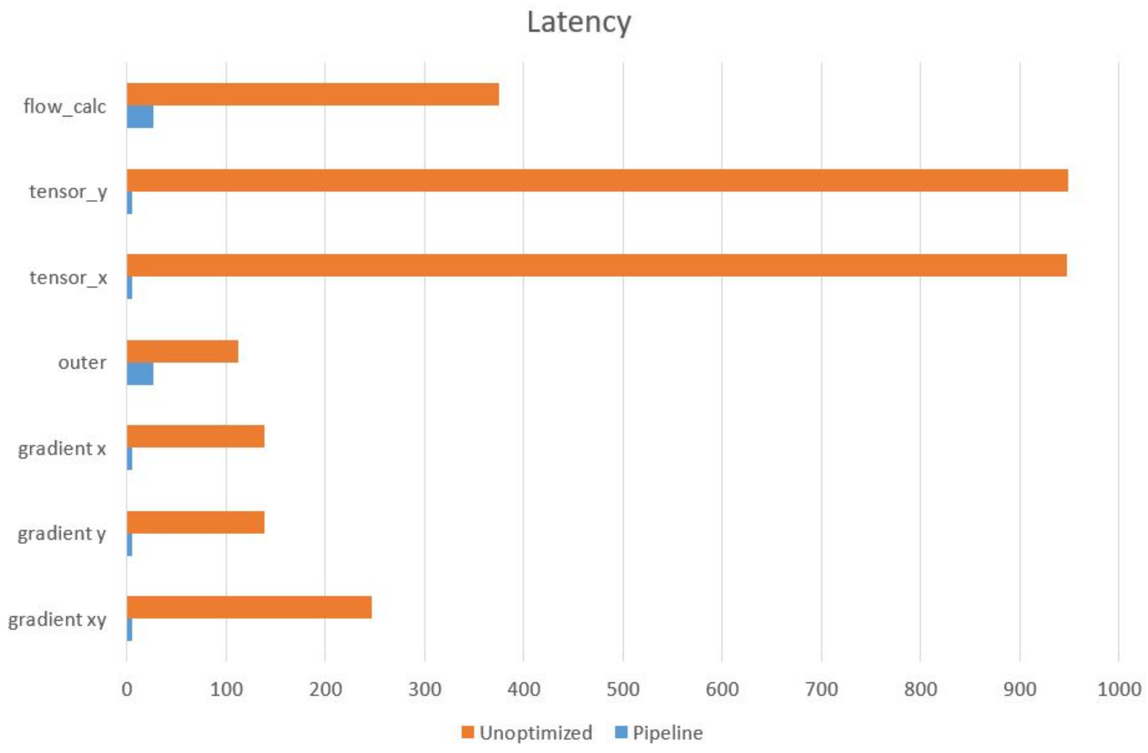
  more efficient communication

```
gradient_xy_calc(frame3_a, gradient_x, gradient_y);
gradient_z_calc(frame1_a, frame2_a, frame3_b, frame4_a, frame5_a, gradient_z);
gradient_weight_y(gradient_x, gradient_y, gradient_z, y_filtered);
gradient_weight_x(y_filtered, filtered_gradient);
outer_product(filtered_gradient, out_product);
tensor_weight_y(out_product, tensor_y);
tensor_weight_x(tensor_y, tensor);
flow_calc(tensor, outputs);
```

```
#pragma HLS STREAM variable=gradient_x depth=default_depth
#pragma HLS STREAM variable=gradient_y depth=default_depth
#pragma HLS STREAM variable=gradient_z depth=max_width*4
#pragma HLS STREAM variable=y_filtered depth=default_depth
#pragma HLS STREAM variable=filtered_gradient depth=default_depth
#pragma HLS STREAM variable=out_product depth=default_depth
#pragma HLS STREAM variable=tensor_y depth=default_depth
#pragma HLS STREAM variable=tensor depth=default_depth
```

Timing & Resource Utilization

Pipelined Latency



Pipeline Optimization

```
+ Latency:
    * Summary:
    +----------+-----------+-----------+-----------+-----------+-----------+---------+
    |  Latency (cycles)    |   Latency (absolute)  |        Interval       | Pipeline|
    |   min    |    max    |    min    |    max    |    min    |    max    |  Type   |
    +----------+-----------+-----------+-----------+-----------+-----------+---------+
    |  76958050|  280098074| 0.770 sec | 2.801 sec |   76958050|  280098074|   none  |
    +----------+-----------+-----------+-----------+-----------+-----------+---------+

+ Latency:
    * Summary:
    +---------+---------+-----------+-----------+---------+---------+---------+
    |  Latency (cycles) |   Latency (absolute)  |      Interval     | Pipeline|
    |   min   |   max   |    min    |    max    |   min   |   max   |  Type   |
    +---------+---------+-----------+-----------+---------+---------+---------+
    |  4027072|  4027072| 40.271 ms | 40.271 ms |  4027072|  4027072|   none  |
    +---------+---------+-----------+-----------+---------+---------+---------+
```

Dataflow Optimization

+ Latency:
  * Summary:

Latency (cycles) min	Latency (cycles) max	Latency (absolute) min	Latency (absolute) max	Interval min	Interval max	Pipeline Type
4027072	4027072	40.271 ms	40.271 ms	4027072	4027072	none

+ Latency:
  * Summary:

Latency (cycles) min	Latency (cycles) max	Latency (absolute) min	Latency (absolute) max	Interval min	Interval max	Pipeline Type
449645	449645	4.496 ms	4.496 ms	449544	449544	dataflow

Stream Optimization

```
+ Latency:
    * Summary:
    +---------+---------+----------+----------+---------+---------+----------+
    |  Latency (cycles) |  Latency (absolute) |       Interval    | Pipeline |
    |   min   |   max   |    min   |    max   |   min   |   max   |   Type   |
    +---------+---------+----------+----------+---------+---------+----------+
    |  4030144|  4030144| 40.301 ms| 40.301 ms|   899081|   899081| dataflow |
    +---------+---------+----------+----------+---------+---------+----------+
```

```
+ Latency:
    * Summary:
    +---------+---------+----------+----------+---------+---------+----------+
    |  Latency (cycles) |  Latency (absolute) |       Interval    | Pipeline |
    |   min   |   max   |    min   |    max   |   min   |   max   |   Type   |
    +---------+---------+----------+----------+---------+---------+----------+
    |   449645|   449645|  4.496 ms|  4.496 ms|   449544|   449544| dataflow |
    +---------+---------+----------+----------+---------+---------+----------+
```

Timming

Pragma Type	Latency (Absolute Max)	Timming(100 MHz)	(200 MHz)
None	1.169 (sec)	8.690 ns	
Pipeline	40.271(ms)	8.657 ns	
w/o Stream	35.806 (ms)	8.510 ns	
w/o Datapack	26.851 (ms)	8.657 ns	
Optimization	4.496 (ms)	8.657 ns	5.527

Compare

Utilization	Unoptimized	Optimization	Available
BRAM	16770	135	280
DPS48E	78	196	220
FF	5388	33066	106400
LUT	7376	26023	53200

BRAM Utilization

* Memory:

Memory	Module	BRAM_18K	FF	LUT	URAM	Words	Bits	Banks	W*Bits*Banks
frame1_a_V_U	optical_flow_framEe0	256	0	0	0	446464	8	2	7143424
frame2_a_V_U	optical_flow_framEe0	256	0	0	0	446464	8	2	7143424
frame3_a_V_U	optical_flow_framEe0	256	0	0	0	446464	8	2	7143424
frame4_a_V_U	optical_flow_framEe0	256	0	0	0	446464	8	2	7143424
frame5_a_V_U	optical_flow_framEe0	256	0	0	0	446464	8	2	7143424
gradient_x_V_U	optical_flow_gradJfO	1024	0	0	0	446464	32	2	28573696
gradient_y_V_U	optical_flow_gradJfO	1024	0	0	0	446464	32	2	28573696
filtered_gradient_x_s_U	optical_flow_gradJfO	1024	0	0	0	446464	32	2	28573696
filtered_gradient_y_s_U	optical_flow_gradJfO	1024	0	0	0	446464	32	2	28573696
filtered_gradient_z_s_U	optical_flow_gradJfO	1024	0	0	0	446464	32	2	28573696
gradient_z_V_U	optical_flow_gradLf8	1024	0	0	0	446464	32	2	28573696
y_filtered_x_V_U	optical_flow_gradLf8	1024	0	0	0	446464	32	2	28573696
y_filtered_y_V_U	optical_flow_gradLf8	1024	0	0	0	446464	32	2	28573696
y_filtered_z_V_U	optical_flow_gradLf8	1024	0	0	0	446464	32	2	28573696
out_product_val_V_U	optical_flow_out_Shg	6112	382	0	0	446464	191	2	170549248
tensor_y_val_V_U	optical_flow_tensThq	6144	384	0	0	446464	192	2	171442176
tensor_val_V_U	optical_flow_tensThq	6144	384	0	0	446464	192	2	171442176
Total		28896	1150	0	0	7589888	903	34	806313984

================================
== Utilization Estimates
================================

* Summary:

Name	BRAM_18K
DSP	-
Expression	-
FIFO	-
Instance	72
Memory	28896
Multiplexer	-
Register	-
Total	28968
Available	280
Utilization (%)	10345

Stream FIFO

* FIFO:

Name	BRAM_18K	FF	LUT	URAM	Depth	Bits	Size:D*B
filtered_gradient_x_s_U	2	66	0	-	1024	32	32768
filtered_gradient_y_s_U	2	66	0	-	1024	32	32768
filtered_gradient_z_s_U	2	66	0	-	1024	32	32768
frame1_a_V_U	1	43	0	-	1024	8	8192
frame2_a_V_U	1	43	0	-	1024	8	8192
frame3_a_V_U	1	43	0	-	1024	8	8192
frame3_b_V_U	1	43	0	-	1024	8	8192
frame4_a_V_U	1	43	0	-	1024	8	8192
frame5_a_V_U	1	43	0	-	1024	8	8192
gradient_x_V_U	2	66	0	-	1024	32	32768
gradient_y_V_U	2	66	0	-	1024	32	32768
gradient_z_V_U	8	90	0	-	4096	32	131072
out_product_val_V_U	11	225	0	-	1024	191	195584
tensor_val_V_U	11	226	0	-	1024	192	196608
tensor_y_val_V_U	11	226	0	-	1024	192	196608
y_filtered_x_V_U	2	66	0	-	1024	32	32768
y_filtered_y_V_U	2	66	0	-	1024	32	32768
y_filtered_z_V_U	2	66	0	-	1024	32	32768
Total	63	1553	0	0	21504	911	1031168

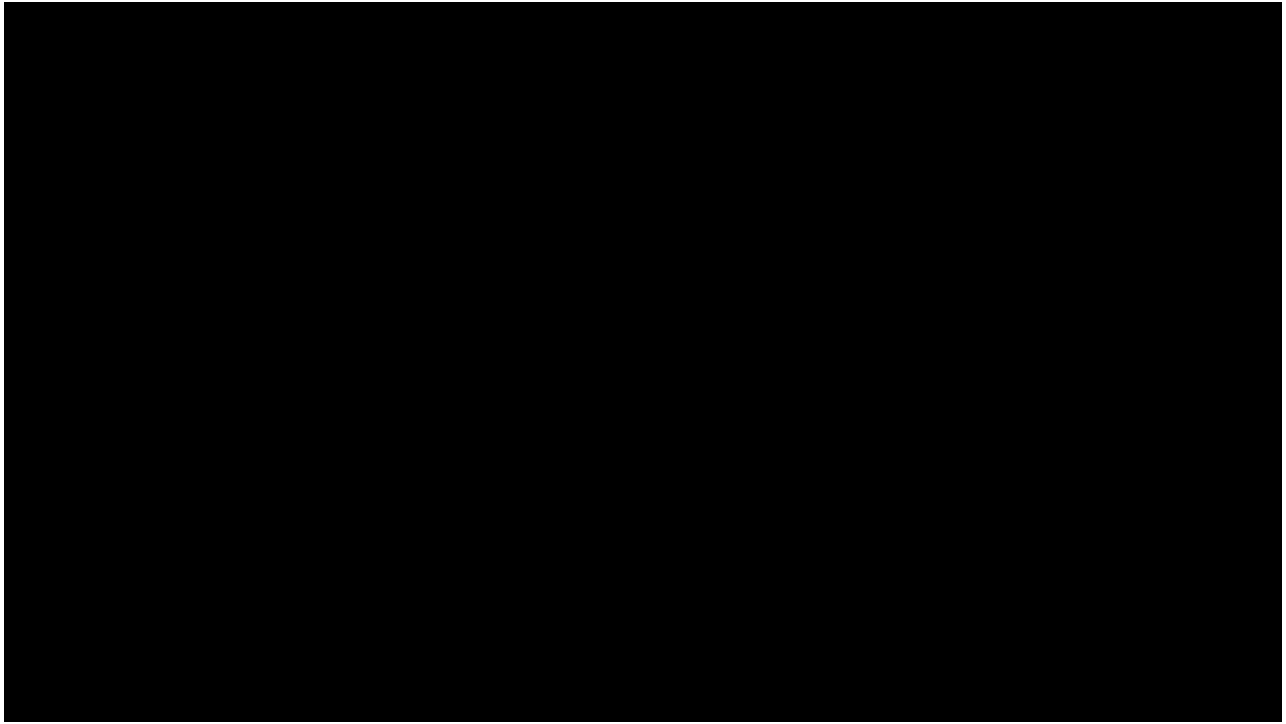
Stream FIFO

```
================================================================
== Utilization Estimates
================================================================
* Summary:
+-----------------+---------+-------+--------+-------+-----+
|       Name      | BRAM_18K| DSP48E|   FF   |  LUT  | URAM|
+-----------------+---------+-------+--------+-------+-----+
|DSP              |        -|      -|       -|      -|    -|
|Expression       |        -|      -|       0|      4|    -|
|FIFO             |       63|      -|    1553|   2355|    -|
|Instance         |       72|    196|   31656|  24073|    0|
|Memory           |        -|      -|       -|      -|    -|
|Multiplexer      |        -|      -|       -|      -|    -|
|Register         |        -|      -|       -|      -|    -|
+-----------------+---------+-------+--------+-------+-----+
|Total            |      135|    196|   33209|  26432|    0|
+-----------------+---------+-------+--------+-------+-----+
|Available        |      280|    220|  106400|  53200|    0|
+-----------------+---------+-------+--------+-------+-----+
|Utilization (%)  |       48|     89|      31|     49|    0|
+-----------------+---------+-------+--------+-------+-----+
```

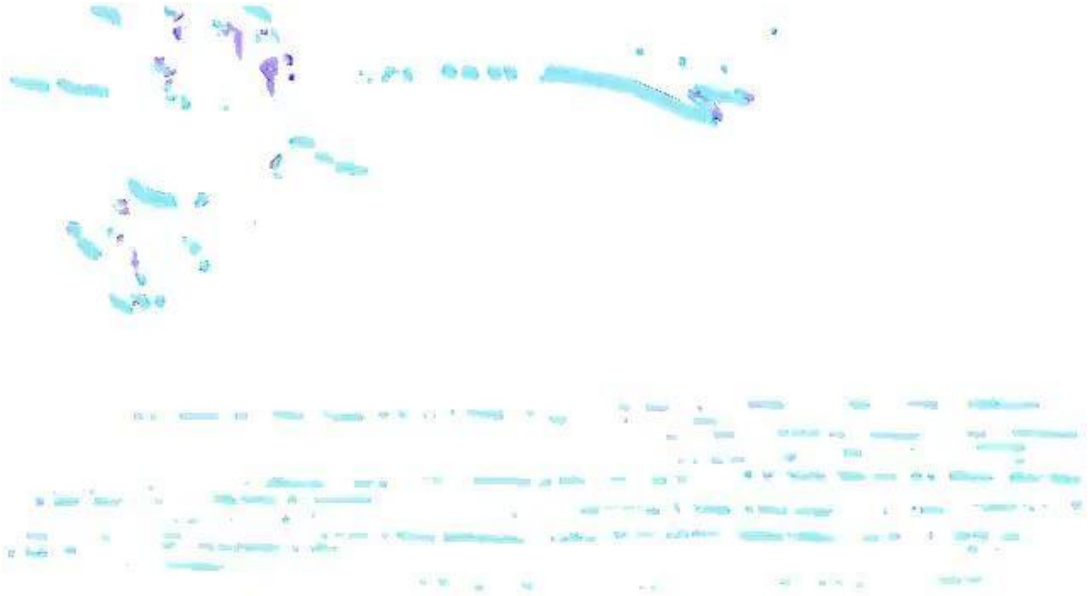
Resource

func/Utlis	BRAM	DSP48E	FF	LUT
gradient_xy	8	8	835	1234
gradient_y	42	42	2981	2200
gradient_x	0	42	2203	1936
outer_product	0	6	333	190
tensor_x	0	36	1923	1748
tensor_y	22	36	1960	1649
flow_calc	0	24	21050	14358

Result



- 31.2 ms / frame
- Blue : x direction
- Purple : y direction



Reference

[1] Wiki - Lucas–Kanade method

[2] Rosetta: A Realistic High-Level Synthesis Benchmark Suite for Software Programmable FPGAs, 2018

[3] Optical Flow Estimation

[4] Introduction To Optical Flow

Github Link

https://github.com/yuweitt/Final-Project-Optical-Flow