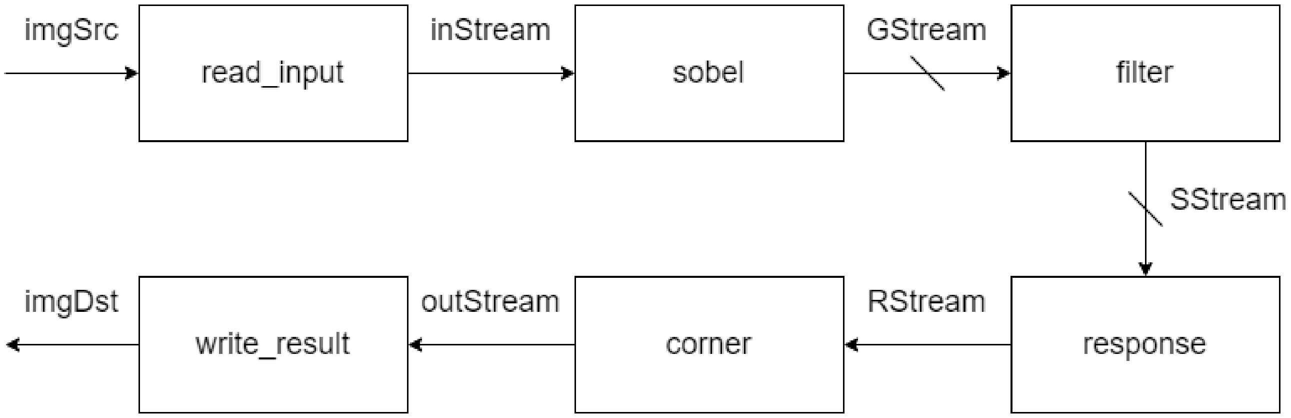
LabC-harris

GIEE b07901073 吳秉軒
GIEE r11943043 潘奕亘
GIEE r11943012 曾維雋

A. Introduction

In this problem, we are going to use HLS to implement hardware-friendly Harris affine detector. The target of this problem is performance. Therefore, we should try to improve the utilization rate and maximize the throughput of the algorithm.

B. Overall System



There are four computing steps in Harris algorithm: sobel operator, box filter, response, and local maximum.

We use Dataflow to implement data movement. At the beginning and the end of computing, there are additional blocks to transform between array and stream data type. Other kernels then use stream to receive and send data.

C. Design Flow

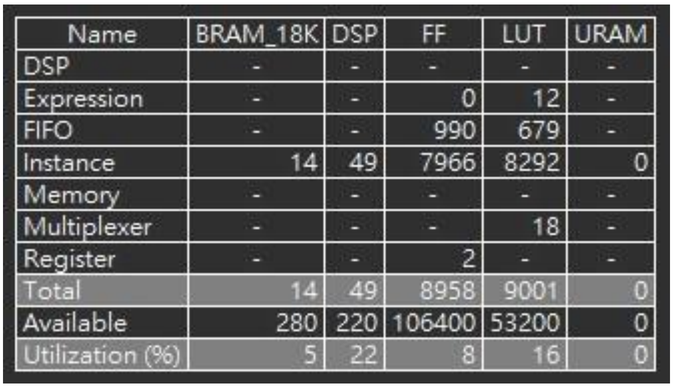
1. Data flow implementation

First, since we use dataflow and expect to receive data from host once the kernels are ready, we use pragma, INTEFACE, to set the block protocol as ap_ctrl_chain.

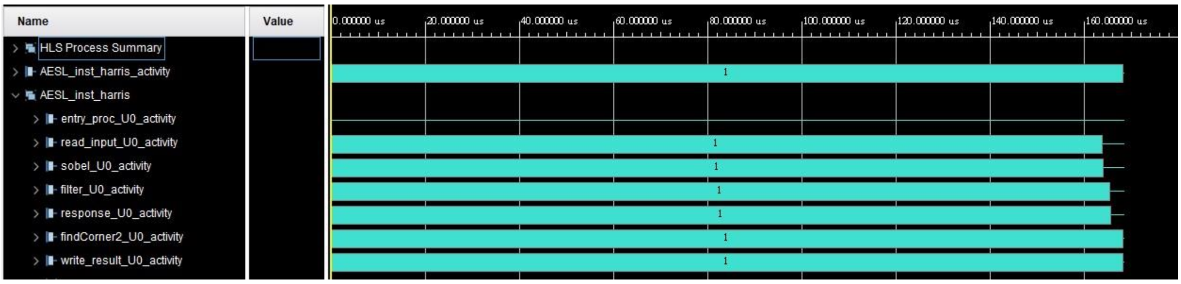
Second, because the input has been transformed from 2D-array to 1D-array, it is easy to read data from input array to stream. Also, to take to most advantage to data-driven design, we set each computing kernel's port as ap_ctrl_none.

Last, since most of computing steps required 2D-window calculations, each kernel needs to create buffer to store their input. Furthermore, to let Internal Interval (II) equals to 1, we use pragma, ARRAY_RESHAPE, to increase bandwidth for each buffer.

The screenshots showed below are utilization report and co-sim timeline:



Name	BRAM_18K	DSP	FF	LUT	URAM
DSP	-	-	-	-	-
Expression	-	-	0	12	-
FIFO	-	-	990	679	-
Instance	14	49	7966	8292	0
Memory	-	-	-	-	-
Multiplexer	-	-	-	18	-
Register	-	-	2	-	-
Total	14	49	8958	9001	0
Available	280	220	106400	53200	0
Utilization (%)	5	22	8	16	0



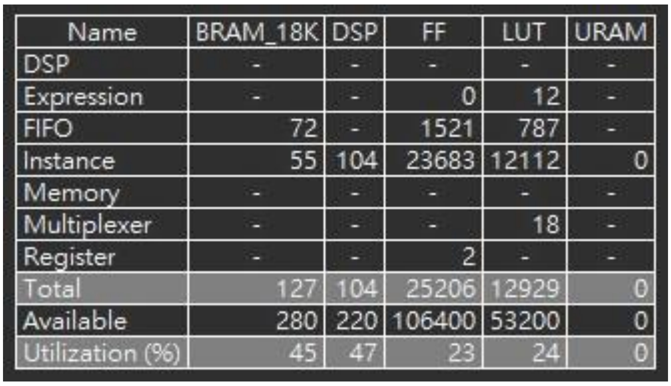
Due to dataflow, all the kernels are active all the time during execution. However, the resource utilization rate is quite low, which means we should use more hardware resources to improve throughput.

2. Bandwidth increasement

We think the easiest and most effective way to utilize more resources is increase bandwidth and create more parallel computing units.

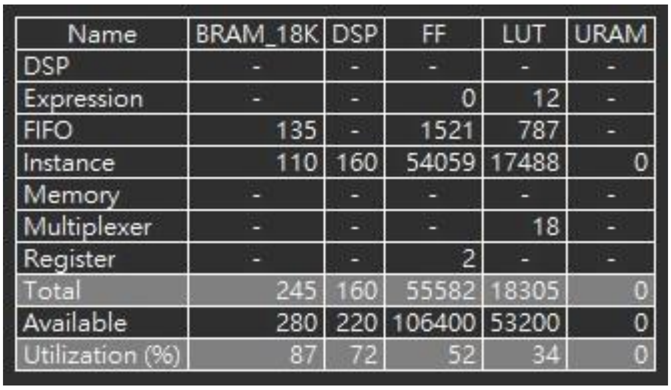
Since the first version has already cost 22% of DSP, we first try to increase the bandwidth to 4. To doing so, we need to modify three regions in our code: use ARRAY_RESHAPE pragma at input, use struct as stream's data type to carry more data, and some indexing problem. To improve scalability, we use parameter, VECTOR_SIZE, to denote the parallelization rate.

The screenshots showed below are utilization report:



Name	BRAM_18K	DSP	FF	LUT	URAM
DSP	-	-	-	-	-
Expression	-	-	0	12	-
FIFO	72	-	1521	787	-
Instance	55	104	23683	12112	0
Memory	-	-	-	-	-
Multiplexer	-	-	-	18	-
Register	-	-	2	-	-
Total	127	104	25206	12929	0
Available	280	220	106400	53200	0
Utilization (%)	45	47	23	24	0

Besides increasing bandwidth, we also improve some operations, which will be explained at next section. Therefore, we can increase bandwidth to 8:



Name	BRAM_18K	DSP	FF	LUT	URAM
DSP	-	-	-	-	-
Expression	-	-	0	12	-
FIFO	135	-	1521	787	-
Instance	110	160	54059	17488	0
Memory	-	-	-	-	-
Multiplexer	-	-	-	18	-
Register	-	-	2	-	-
Total	245	160	55582	18305	0
Available	280	220	106400	53200	0
Utilization (%)	87	72	52	34	0

3. Operation optimization

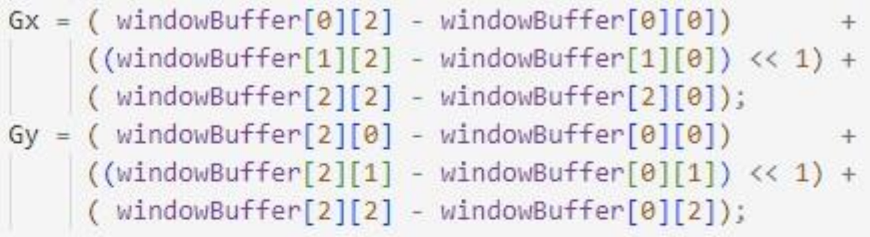
There are several operations we can improve:

a. Sobel operator

Sobel operator is used to compute the gradient of graphs.

$$I_x = \begin{bmatrix} -1 & 0 & +1 \\ -2 & 0 & +2 \\ -1 & 0 & +1 \end{bmatrix}, I_y = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ +1 & +2 & +1 \end{bmatrix}$$

Since I_x, I_y are known at compile time, we can improve this operation with following code instead of bitwise matrix multiplication.



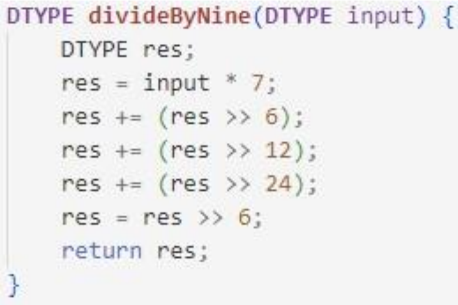
```
Gx = ( windowBuffer[0][2] - windowBuffer[0][0])       +
     ((windowBuffer[1][2] - windowBuffer[1][0]) << 1) +
     ( windowBuffer[2][2] - windowBuffer[2][0]);
Gy = ( windowBuffer[2][0] - windowBuffer[0][0])       +
     ((windowBuffer[2][1] - windowBuffer[0][1]) << 1) +
     ( windowBuffer[2][2] - windowBuffer[0][2]);
```

b. Box filter

There is a constant division in box filter.

$$B = \frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix},$$

Again, since the constant is known at compile time, we can implement constant division with a multiplication and several shift and add.



```
DTYPE divideByNine(DTYPE input) {
    DTYPE res;
    res = input * 7;
    res += (res >> 6);
    res += (res >> 12);
    res += (res >> 24);
    res = res >> 6;
    return res;
}
```

c. Response

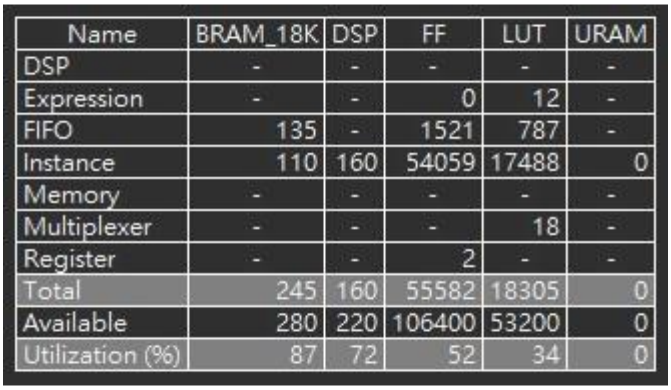
In response calculation, there is an additional parameter, α.

$$R = \det(M) - \alpha \cdot \mathrm{tr}(M)^2$$

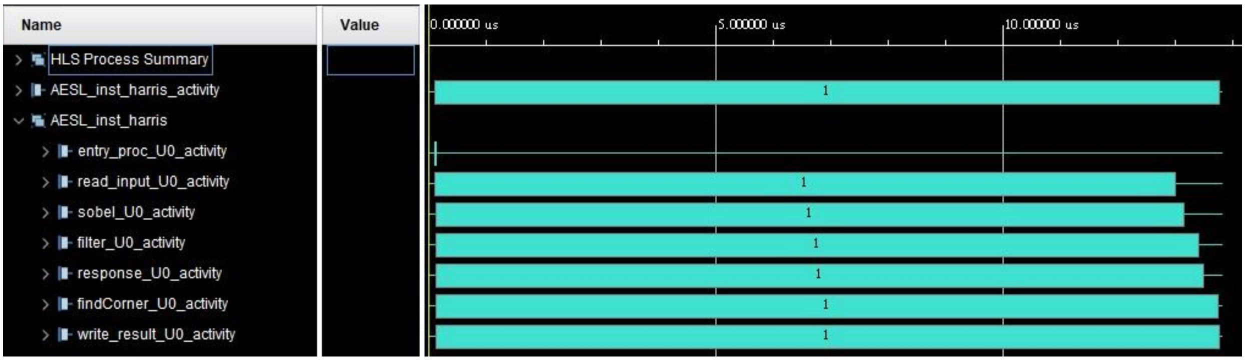
Although α is also an input, we can change its data type from double to arbitrary precision (AP) data type. Since the problem allows a few errors, we can trade precision for simple operation.

D. Performance

The screenshots showed below are utilization report and co-sim timeline of our final design:



Name	BRAM_18K	DSP	FF	LUT	URAM
DSP	-	-	-	-	-
Expression	-	-	0	12	-
FIFO	135	-	1521	787	-
Instance	110	160	54059	17488	0
Memory	-	-	-	-	-
Multiplexer	-	-	-	18	-
Register	-	-	2	-	-
Total	245	160	55582	18305	0
Available	280	220	106400	53200	0
Utilization (%)	87	72	52	34	0



Score:

$$T_{clock} = 6ns, \tau_{simulation} = 13650ns, F_{max} = 228.78MHz, \frac{T_{clock}}{\tau_{simulation}} * F_{max} = 100562$$

E. Code

https://github.com/aahwu/AAHLS_labC_Harris