一、DMA 的存储地址

以 RS_BRIEF 的计算作为实验情况。pl 模块的输入为图片中行优先顺序的像素值，每个像素 16bit，高 8bit 为是否为特征点（全为 0 代表不是特征点），低 8bit 为像素灰度值。输出为 32bit 行坐标，32bit 列坐标，256bit 描述子以及 192bit0（AXI 标准要求 512 位）。

在图片中设置 4964 个特征点，可以看到，pl 运行时间约为 0.020s。

In [1]:
```
from pynq import Overlay
from pynq import Xlnk
import pprint
import matplotlib.pyplot as plt
import numpy as np
import cv2
from time import time
xlnk = Xlnk()
overlay = Overlay('./RS_BRIEF_ultra96-v2.bit')
dma_src = overlay.axi_dma_src
dma_des = overlay.axi_dma_des
dma_des.recvchannel.start()
dma_src.sendchannel.start()
```

In [2]:
```
img_ori = cv2.imread('./test_data/'+'000000'+'.png')
img_gray = cv2.cvtColor(img_ori,cv2.COLOR_BGR2GRAY)
src_buf = xlnk.cma_array(shape=(1241*376,), dtype=np.uint16)
view=np.frombuffer(src_buf,dtype = np.uint16,count = -1)
np.copyto(view,img_gray.ravel(),casting='same_kind')
des_buf = xlnk.cma_array(shape=(8192,8), dtype=np.uint64)
for i in range (1241*20,1241*24):
    src_buf[i] = src_buf[i] + 0b111100000000
```

In [3]:
```
t0 = time()
dma_des.recvchannel.transfer(des_buf)
dma_src.sendchannel.transfer(src_buf)
# t1 = time()
# tmp = 0
# for i in range(10000):
#     tmp = tmp + 1
# t2 = time()
dma_src.sendchannel.wait()
dma_des.recvchannel.wait()
des_buf.flush()
t3 = time()
# print(t1-t0)
# print(t2-t0)
print('time of computing RS BRIEF:'+ str(t3-t0) + 'seconds')
bytes_read=dma_des.mmio.read(0x58)
featurePointsNum = int(bytes_read/64) - 1
print(str(featurePointsNum)+' feature points are detected')
```

```
time of computing RS BRIEF:0.019983291625976562seconds
4964 feature points are detected
```

在 wait 命令前加入 10000 次加法程序。

Transfer 命令耗时约 0.001s，加法程序耗时约为 0.010s，pl 运行时间仍约为 0.020s，两者同时运行

In [3]:
```
t0 = time()
dma_des.recvchannel.transfer(des_buf)
dma_src.sendchannel.transfer(src_buf)
t1 = time()
tmp = 0
for i in range(10000):
    tmp = tmp + 1
t2 = time()
dma_src.sendchannel.wait()
dma_des.recvchannel.wait()
des_buf.flush()
t3 = time()
print('time of transfer:'+ str(t1-t0) + 'seconds')
print('time of addition:'+ str(t2-t0) + 'seconds')
print('time of computing RS BRIEF:'+ str(t3-t0) + 'seconds')
bytes_read=dma_des.mmio.read(0x58)
featurePointsNum = int(bytes_read/64) - 1
print(str(featurePointsNum)+' feature points are detected')
```

```
time of transfer:0.0007495880126953125seconds
time of addition:0.009748935699462891seconds
time of computing RS BRIEF:0.019858121871948242seconds
4964 feature points are detected
```

在 wait 命令前加入 50000 次加法程序，程序耗时约为 0.045s，可以看出 wait 命令几乎没有耗时。

In [3]:
```
t0 = time()
dma_des.recvchannel.transfer(des_buf)
dma_src.sendchannel.transfer(src_buf)
t1 = time()
tmp = 0
for i in range(50000):
    tmp = tmp + 1
t2 = time()
dma_src.sendchannel.wait()
dma_des.recvchannel.wait()
des_buf.flush()
t3 = time()
print('time of transfer:'+ str(t1-t0) + 'seconds')
print('time of addition:'+ str(t2-t0) + 'seconds')
print('time of computing RS BRIEF:'+ str(t3-t0) + 'seconds')
bytes_read=dma_des.mmio.read(0x58)
featurePointsNum = int(bytes_read/64) - 1
print(str(featurePointsNum)+' feature points are detected')
```

```
time of transfer:0.0007622241973876953seconds
time of addition:0.045334815979003906seconds
time of computing RS BRIEF:0.04602861404418945seconds
4964 feature points are detected
```

若确定 ps 侧运行时间大于 pl，可以不使用 wait 命令，一样会获取到结果，因此 DMA 应该是自动把数据储存在 buffer 的内存中。

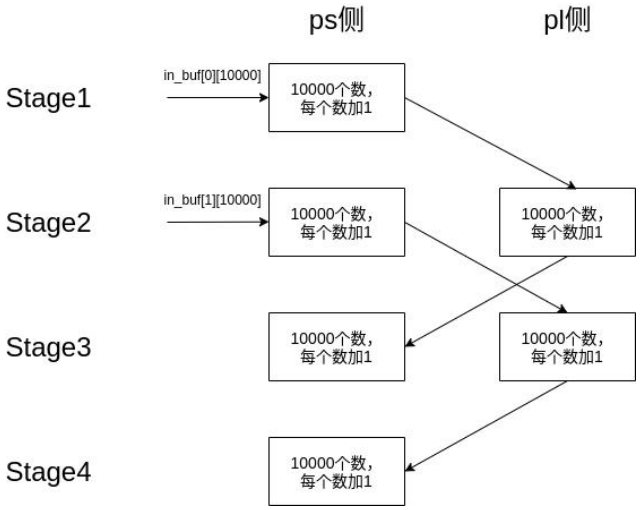
In [3]:
```
t0 = time()
dma_des.recvchannel.transfer(des_buf)
dma_src.sendchannel.transfer(src_buf)
t1 = time()
tmp = 0
for i in range(50000):
    tmp = tmp + 1
t2 = time()
# dma_src.sendchannel.wait()
# dma_des.recvchannel.wait()
des_buf.flush()
t3 = time()
print('time of transfer:'+ str(t1-t0) + 'seconds')
print('time of addition:'+ str(t2-t0) + 'seconds')
print('time of computing RS BRIEF:'+ str(t3-t0) + 'seconds')
bytes_read=dma_des.mmio.read(0x58)
featurePointsNum = int(bytes_read/64) - 1
print(str(featurePointsNum)+' feature points are detected')
```

```
time of transfer:0.0007534027099609375seconds
time of addition:0.04622030258178711seconds
time of computing RS BRIEF:0.04652810096740723seconds
4964 feature points are detected
```

二、用 Multiprocessing 进行 ps 与 pl 并行

相关代码位于~/addition 中。

使用 ps 和 pl 进行 10000 个数每个数加 3 的工作。



首先使用 hls 生成 ip 核，在 vivado 中生成比特流。

引用库以及定义函数。其中 ps_addition(i)为 ps 侧的 in_buf[i][10000]的加法，pl_addition(i)为 pl 侧的 in_buf[i][10000]的加法。

In [1]:
```
from pynq import Overlay
from pynq import Xlnk
import numpy as np
from time import time
import multiprocessing as mp
```

In [2]:
```
def buf_cp(i):
    global in_buf
    global out_buf
    for j in range(10000):
        in_buf[i][j] = out_buf[i][j]

def ps_addition(i):
    t0 = time()
    global in_buf
    for j in range(10000):
        out_buf[i][j] = in_buf[i][j] + 1
    t1 = time()
    print('ps:',(t1-t0),'seconds')
    buf_cp(i)

def pl_addition(i):
    t0 = time()
    global in_buf
    global out_buf
    global dma_in
    global dma_out
    dma_in.sendchannel.transfer(in_buf[i])
    dma_out.recvchannel.transfer(out_buf[i])
    dma_in.sendchannel.wait()
    dma_out.recvchannel.wait()
    out_buf.flush()
    t1 = time()
    print('pl:',(t1-t0),'seconds')
    buf_cp(i)
```

初始化输入数组，in_buf[0]全为 0，inbuf[1]全为 1。

定义 6 个 process 分别运行 ps 与 pl 的加法,并按顺序执行。

```
In [3]: if __name__=='__main__':
            xlnk = Xlnk()
            overlay = Overlay('./addition_ultra96-v2.bit')
            dma_in = overlay.axi_dma_in
            dma_out = overlay.axi_dma_out
            dma_out.sendchannel.start()
            dma_in.recvchannel.start()

            in_buf = xlnk.cma_array(shape=(2,10000), dtype=np.uint8)
            out_buf = xlnk.cma_array(shape=(2,10000), dtype=np.uint8)

            for i in range (2):
                for j in range(10000):
                    in_buf[i][j] = i

            mp_ps_0 = mp.Process(target=ps_addition,args=(0,))
            mp_ps_1 = mp.Process(target=ps_addition,args=(1,))
            mp_ps_2 = mp.Process(target=ps_addition,args=(0,))
            mp_ps_3 = mp.Process(target=ps_addition,args=(1,))

            mp_pl_0 = mp.Process(target=pl_addition,args=(0,))
            mp_pl_1 = mp.Process(target=pl_addition,args=(1,))

            print('Stage1:')
            mp_ps_0.start()
            mp_ps_0.join()

            print('\nStage2:')
            mp_ps_1.start()
            mp_pl_0.start()
            mp_ps_1.join()
            mp_pl_0.join()

            print('\nStage3:')
            mp_ps_2.start()
            mp_pl_1.start()
            mp_ps_2.join()
            mp_pl_1.join()

            print('\nStage4:')
            mp_ps_3.start()
            mp_ps_3.join()

            print('\n')

            for i in range(2):
                print([x for x in in_buf[i]])

            in_buf.freebuffer()
            out_buf.freebuffer()
            dma_in.sendchannel.stop()
            dma_out.recvchannel.stop()
```

输出结果正确。运行时间情况如下。

```
Stage1:
ps: 1.4982678890228271 seconds

Stage2:
pl: 0.0015506744384765625 seconds
ps: 1.5326566696166992 seconds

Stage3:
pl: 0.001462697982788086 seconds
ps: 1.5111794471740723 seconds

Stage4:
ps: 1.518165111541748 seconds


[3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3,
3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3,
3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3,
3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3,
```

三、并行程序运行时间测法

先测两程序单独运行分别所需要的时间，再测试一起运行所需要的总时间，最后进行比较。

分别运行所需时间：

In [3]:
```
def pl(q):
    global dma_des
    global dma_src
    global des_buf
    global src_buf
    dma_des.recvchannel.transfer(des_buf)
    dma_src.sendchannel.transfer(src_buf)
    dma_src.sendchannel.wait()
    dma_des.recvchannel.wait()
    des_buf.flush()
    t_pl = time()
    q.put(t_pl)

def ps(q):
    t0 = time()
    tmp = 0
    for i in range(500000):
        tmp = tmp + 1
    t_ps = time()
    q.put(t_ps)

q_ps=mp.Queue();
q_pl=mp.Queue();
mp_ps = mp.Process(target=ps,args=(q_ps,))
mp_pl = mp.Process(target=pl,args=(q_pl,))

t0=time()
mp_ps.start()
mp_ps.join()
print('ps:',q_ps.get()-t0,'seconds')

t0=time()
mp_pl.start()
mp_pl.join()
print('pl:',q_pl.get()-t0,'seconds')
```

```
ps: 0.2768585681915283 seconds
pl: 0.03991818428039551 seconds
```

并行所需时间：

In [4]:
```
def pl(q):
    global dma_des
    global dma_src
    global des_buf
    global src_buf
    dma_des.recvchannel.transfer(des_buf)
    dma_src.sendchannel.transfer(src_buf)
    dma_src.sendchannel.wait()
    dma_des.recvchannel.wait()
    des_buf.flush()
    t_pl = time()
    q.put(t_pl)

def ps(q):
    t0 = time()
    tmp = 0
    for i in range(500000):
        tmp = tmp + 1
    t_ps = time()
    q.put(t_ps)


q=mp.Queue();
mp_ps = mp.Process(target=ps,args=(q,))
mp_pl = mp.Process(target=pl,args=(q,))

t0=time()
mp_ps.start()
mp_pl.start()
mp_ps.join()
mp_pl.join()

t1 = time()
print('total:',t1-t0,'seconds')
bytes_read=dma_des.mmio.read(0x58)
featurePointsNum = int(bytes_read/64) - 1
print(str(featurePointsNum)+' feature points are detected')
```

```
total: 0.290283203125 seconds
4964 feature points are detected
```