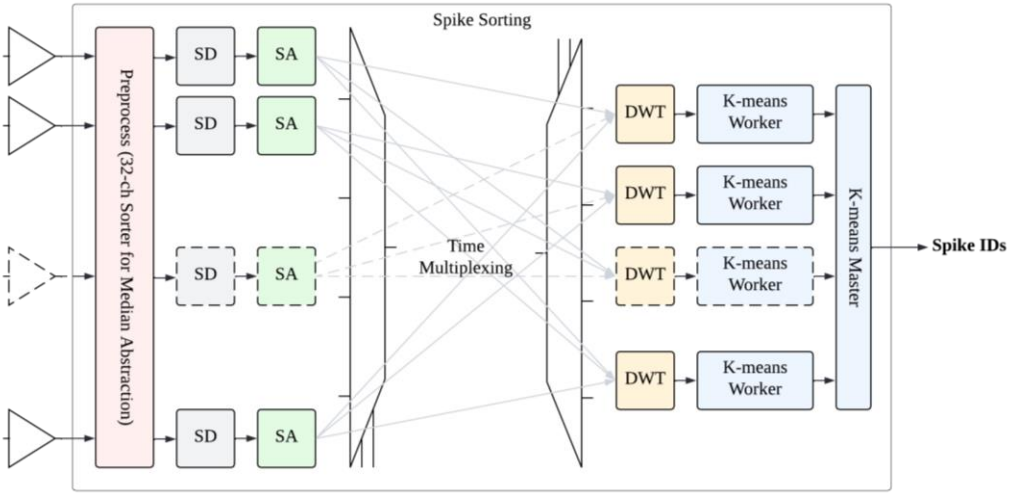
11020EE521800 Application Acceleration with High-Level Synthesis

Final Project – Spike Sorting Acceleration and Beamforming Acceleration

Team Member: 109061564 馬婕芸 110061551 呂易縉 110061540 賴聖耘

I. Spike Sorting Acceleration – Introduction

Spike sorting (SS) is the grouping of action potential (AP) into clusters based on the similarity of their shapes, and it is crucial step to extract information from extracellular recordings. With an emerging generation of high-density microelectrode arrays (MEAs) capable of recording spiking activity from thousands of neurons, reliable and scalable spike detection and analysis are needed. Historically, **spike sorting algorithm including spike detection (SD), spike alignment (SA), feature extraction (FE) and classification**. The whole SS system diagram is shown below.



Among these steps, the bottleneck is feature extraction and classification, discrete wavelet transform (DWT) and k-means clustering (K-means) in our system, since it needs massive computation resources. As a result, we **propose to design an end-to-end acceleration application for these two steps using Vitis tool**. As for SD and SA, we use MATLAB to do simulation and generate input data for DWT kernel. In this way, we can explore:

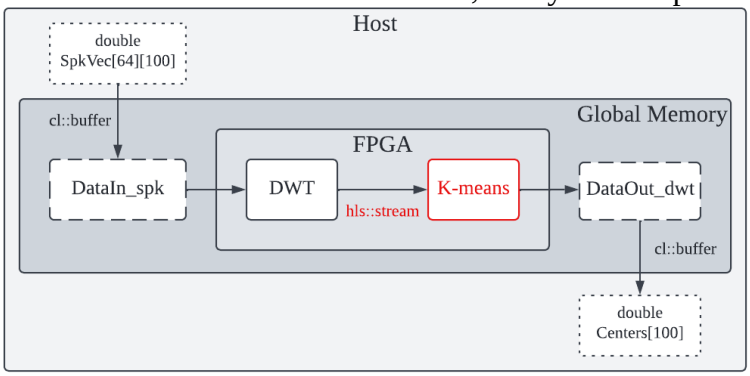
i. Whether the proposed system is feasible or not?
ii. Is K-means algorithm provided by Vitis library scalable or not?
iii. How many resources will each kernel take?
iv. What is the throughput of whole the SS system?

II. Spike Sorting Acceleration – SS System Description

The proposed system for DWT and K-means is shown below. SpkVec[][] is spike data, and it is preprocessed (detected and aligned) by MATLAB. Each spike has 64 points sampled at 32kHz, and each point is `double`. The input will then be buffered in global memory and serve as input for a DWT kernel. DWT kernel will do 4-layer convolution with an 8-point sym-4 wavelet shown below.

```
double HiD[8] = {-0.0322, -0.0126, 0.0992, 0.2979, -0.8037, 0.4976, 0.0296, -0.0758};
double LoD[8] = {-0.0758, -0.0296, 0.4976, 0.8037, 0.2979, -0.0992, -0.0126, 0.0322};
```

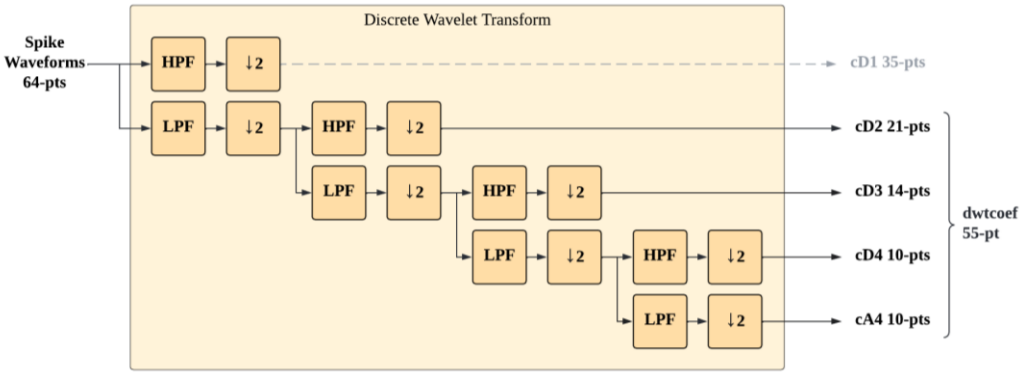
The result is 55-point DWT coefficient, and each point is `double`. And they will be streamed into K-means kernel. Its output is trained center of each clusters, and it will be copied in global memory again. In our system, we assumed there are 100 clusters. As a result, the system output is Centers[].



However, the whole system is not successfully built, since K-means is not applicable for spike data set. Also, the interface of each sub-function in Vitis K-means kernel is hard to modify. Alternatively, we built end-to-end acceleration application for DWT and K-means kernel respectively in order to make sure each kernel is feasible.

III. Spike Sorting Acceleration – DWT System Description

The system including host program, buffers for I/O (I: spk, O: dwt) in global memory and dwt function. The dwt function scheme is using lifting scheme shown below. Since each layer is doing the same thing, the whole function can be pipelined to enhance the latency.



IV. Spike Sorting Acceleration – K-means System Description

K-means clustering algorithm is unsupervised. The step-by-step details is shown below:

Algorithm 1 k-means algorithm

1: Specify the number k of clusters to assign.
2: Randomly initialize k centroids.
3: **repeat**
4: **expectation:** Assign each point to its closest centroid.
5: **maximization:** Compute the new centroid (mean) of each cluster.
6: **until** The centroid positions do not change.

Thanks to Vitis Libarary, it provides the HLS implementation of k-means clustering algorithm. Originally, it supports IRIS dataset (150 samples) with 4 features clustered into 3 clusters. The data format is shown below:

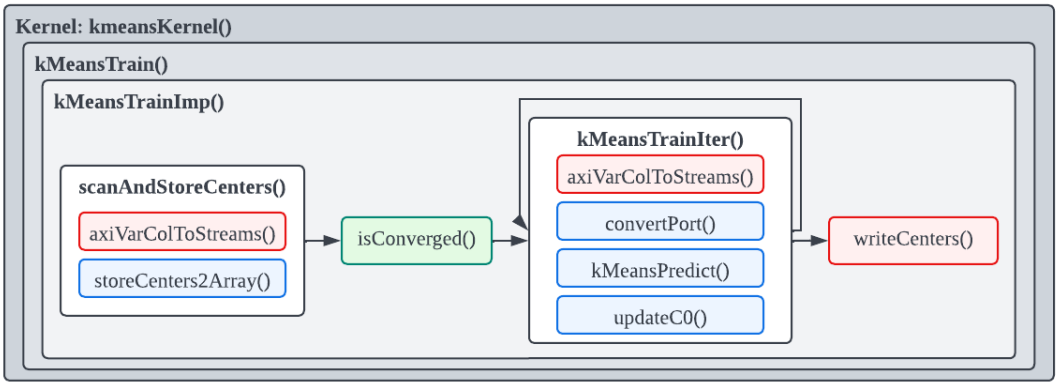
```
float irisVec[150][5] = {
    {5.1, 3.5, 1.4, 0.2, 0}, {4.9, 3.0, 1.4, 0.2, 0}, {4.7, 3.2, 1.3, 0.2, 0}, {4.6, 3.1, 1.5, 0.2, 0},
    {5.0, 3.6, 1.4, 0.2, 0}, {5.4, 3.9, 1.7, 0.4, 0}, {4.6, 3.4, 1.4, 0.3, 0}, {5.0, 3.4, 1.5, 0.2, 0},
    {4.4, 2.9, 1.4, 0.2, 0}, {4.9, 3.1, 1.5, 0.1, 0}, {5.4, 3.7, 1.5, 0.2, 0}, {4.8, 3.4, 1.6, 0.2, 0},
    {4.8, 3.0, 1.4, 0.1, 0}, {4.3, 3.0, 1.1, 0.1, 0}, {5.8, 4.0, 1.2, 0.2, 0}, {5.7, 4.4, 1.5, 0.4, 0},
    {5.4, 3.9, 1.3, 0.4, 0}, {5.1, 3.5, 1.4, 0.3, 0}, {5.7, 3.8, 1.7, 0.3, 0}, {5.1, 3.8, 1.5, 0.3, 0},
    {5.4, 3.4, 1.7, 0.2, 0}, {5.1, 3.7, 1.5, 0.4, 0}, {4.6, 3.6, 1.0, 0.2, 0}, {5.1, 3.3, 1.7, 0.5, 0},
    {4.8, 3.4, 1.9, 0.2, 0}, {5.0, 3.0, 1.6, 0.2, 0}, {5.0, 3.4, 1.6, 0.4, 0}, {5.2, 3.5, 1.5, 0.2, 0},
    {5.2, 3.4, 1.4, 0.2, 0}, {4.7, 3.2, 1.6, 0.2, 0}, {4.8, 3.1, 1.6, 0.2, 0}, {5.4, 3.4, 1.5, 0.4, 0},
```

In 32-channel SS system, the dwt coefficient has 55 features. Also, based on academic papers, each channel usually has 3~4 clusters. As a result, we set cluster number = 100. And the data format is also shown below:

```
double dwtVec[100][55] = {
{-7.0235, 3.8410, -4.5762, 4.6592, 9.1547, -13.1232, -2.6128, -0.1984, 1.0497, 20.0001, -21.5950, 5.7262, 18.188
{-2.4417, -0.6240, 2.3464, 14.7158, 19.4765, -17.5993, 25.7181, -9.8446, 36.0267, 3.3977, -16.3810, -26.4512, 7.
{5.2124, 14.2644, -28.3224, 21.1945, -21.3872, 5.9805, -23.1323, -27.1067, 25.9398, 8.0974, 17.5096, -0.1240, 45
{-32.2899, -2.9580, 11.4581, 34.1381, 3.3194, -16.3810, -26.4512, 7.1154, -29.2999, -20.7015, 21.0519, -18.3898,
{-24.2525, -4.4783, 12.5589, 6.8763, -16.2969, -26.4512, 7.1154, -29.2999, -20.7015, 21.0519, -18.3898, -21.3633
{-49.7440, -18.5249, 37.0437, 42.4505, -17.6086, 23.2845, -12.8342, -23.0943, -4.4169, 36.9135, -14.1872, -0.784
{-13.6055, -10.2320, 24.0706, -10.6271, -28.0042, 20.6874, -19.1736, 0.6237, -12.1094, -7.2492, 40.4966, -20.459
{10.4628, -5.3713, 5.9504, -11.7405, -34.2474, 24.7603, -6.7700, -11.2996, -26.5054, -24.1523, 24.2371, -52.4138
```

Here is the k-means system diagram. The k-means system including host program, DDR and kmeanskernel function. The interface is quite complicated. The blue blocks use HLS stream to communicate, and red blocks communicate with external DDR. The input of this function is data samples

with features, and the output is the centers of each clusters. We can then know which cluster a new data sample belongs to.



Each function description is shown below:

- kmeansKernel() is k-means clustering kernel called by host program.
- kmeansTrain() is k-means clustering function.
- kMeansTrainImp() is main HLS implementation of k-means clustering.
  - scanAndStoreCenters() scans data from DDR and store initial centers in local memory.
    - axiVarColToStreams() scans data (DWT coefficients) from DDR.
    - storeCenters2Array() stores centers to local memory.
  - isCoverage() decides whether there is any clusters should be merged.
  - kmeansTrainIter() is main k-means clustering function in each iteration.
    - axiVarColToStreams() scans data (DWT coefficients) from DDR.
    - convertPort() changes input data into various streams.
    - kMeansPredict() decides which cluster a new data sample belongs.
    - updateC0() update each cluster's center.
  - writeCenters() writes the centers back into DDR.

V. Spike Sorting Acceleration – Results & Discussion

1. DWT system:

i. Hardware Target: Resources

We can find that a single DWT kernel only takes up 3.76% LUT, 0.6% BRAM and 3.06% DSP in FPGA. The U50 platform may not suitable for up to 32 channels, since $100/3.76 \approx 27$.

Name	LUT	LUTAsMem	REG	BRAM	URAM	DSP
Platform	99966	8343	123023	178	0	4
User Budget	770050	393673	1620337	1166	640	5936
Used Resources	28496	3751	33888	8	0	182
Unused Resources	741554	389922	1586449	1158	640	5754
dwtkernel (1)	28496	3751	33888	8	0	182
dwtkernel_1	28496	3751	33888	8	0	182

ii. Hardware Target: Latency

We can find that the dwtkernel takes 4.070 us to generate data. The latency is much shorter than software simulation. After pipelining, the throughput can reach II=1.

Latency Information

Compute Unit	Kernel Name	Module Name	Start Interval	Best (cycles)	Avg (cycles)	Worst (cycles)	Best (absolute)
dwtkernel_1	dwtkernel	dwtkernel_Pipeline_VITIS_LOOP_45_1	87	87	87	87	0.290 us
dwtkernel_1	dwtkernel	dwtkernel_Pipeline_VITIS_LOOP_61_2	253	253	253	253	0.843 us
dwtkernel_1	dwtkernel	dwtkernel_Pipeline_VITIS_LOOP_82_3	9	9	9	9	29.997 ns
dwtkernel_1	dwtkernel	dwtkernel_Pipeline_VITIS_LOOP_99_4	80	80	80	80	0.267 us
dwtkernel_1	dwtkernel	dwtkernel_Pipeline_VITIS_LOOP_124_5	9	9	9	9	29.997 ns
dwtkernel_1	dwtkernel	dwtkernel_Pipeline_VITIS_LOOP_140_6	74	74	74	74	0.247 us
dwtkernel_1	dwtkernel	dwtkernel_Pipeline_VITIS_LOOP_165_7	9	9	9	9	29.997 ns
dwtkernel_1	dwtkernel	dwtkernel_Pipeline_VITIS_LOOP_181_8	71	71	71	71	0.237 us
dwtkernel_1	dwtkernel	dwtkernel_Pipeline_VITIS_LOOP_201_9	92	92	92	92	0.307 us
dwtkernel_1	dwtkernel	dwtkernel_Pipeline_VITIS_LOOP_202_10	85	85	85	85	0.283 us
dwtkernel_1	dwtkernel	dwtkernel_Pipeline_VITIS_LOOP_203_11	81	81	81	81	0.270 us
dwtkernel_1	dwtkernel	dwtkernel_Pipeline_VITIS_LOOP_204_12	81	81	81	81	0.270 us
dwtkernel_1	dwtkernel	dwtkernel	1222	1221	1221	1221	4.070 us

iii. Hardware Target: Co-simulation

The co-simulation is successfully in target hardware stage.

```
Found Platform
Platform Name: Xilinx
INFO: Reading /users/course/2022S
Loading: '/users/course/2022S/HLS
hw: 0.00761115/ sw: 0.00761115
hw: -1.0289/ sw: -1.0289
hw: 1.40291/ sw: 1.40291
hw: -0.251538/ sw: -0.251538
hw: -0.00551868/ sw: -0.00551868
hw: 0.00084852/ sw: 0.00084852
hw: 0.00084852/ sw: 0.00084852
hw: 0.00084852/ sw: 0.00084852
hw: 0.00084852/ sw: 0.00084852
hw: 0.00084852/ sw: 0.00084852
hw: 0.00084852/ sw: 0.00084852
hw: 0.00084852/ sw: 0.00084852
hw: 0.00084852/ sw: 0.00084852
hw: 0.00084852/ sw: 0.00084852
hw: 0.00084852/ sw: 0.00084852
hw: 0.0404717/ sw: 0.0404717
hw: 0.358819/ sw: 0.358819
hw: -1.36879/ sw: -1.36879
hw: 1.18532/ sw: 1.18532
hw: -0.322907/ sw: -0.322907
hw: 1.6798/ sw: 1.6798
hw: -1.30458/ sw: -1.30458
```

2. K-means system:

i. Hardware Target: Resources

We can find that a single k-means kernel only takes up 21% LUT, 2% BRAM and 1% DSP in FPGA. The U50 platform may not suitable for up to 32 channels, since $100/21 \approx 5$. Compared to DWT system, K-means takes up much larger resources. The reason maybe the function is pipelined and paralleled to each high throughput. However, SS system does not need this characteristic.

Name	BRAM_18K	DSP	FF	LUT	URAM
DSP	-	-	-	-	-
Expression	-	-	0	240	-
FIFO	-	-	-	-	-
Instance	60	64	60712	185161	0
Memory	-	-	1	77	10
Multiplexer	-	-	-	1240	-
Register	-	-	707	-	-
Total	60	64	61420	186718	10
Available SLR	1344	2976	871680	435840	320
Utilization SLR (%)	4	2	7	42	3
Available	2688	5952	1743360	871680	640
Utilization (%)	2	1	3	21	1

ii. Hardware Target: Latency

Latency Information

Compute Unit	Kernel Name	Module Name	Start Interval	Best (cycles)
kmeansKernel	kmeansKernel	entry_proc	0	0
kmeansKernel	kmeansKernel	readRaw_512_32_64_Pipeline_READ_RAW	600003	600003
kmeansKernel	kmeansKernel	readRaw_512_32_64_s	4 ~ 600078	4
kmeansKernel	kmeansKernel	cageShitRight_512_64_s	1	0
kmeansKernel	kmeansKernel	varSplit_512_64_Pipeline_LOOP1_LOOP2	600005	600005
kmeansKernel	kmeansKernel	varSplit_512_64_s	600009	600009
kmeansKernel	kmeansKernel	axiVarColToStreams_32_512_64_s	600010 ~ 600079	600083
kmeansKernel	kmeansKernel	dupStrm_64_8_7_Pipeline_VITIS_LOOP_184_1	70002	70002
kmeansKernel	kmeansKernel	dupStrm_64_8_7_Pipeline_VITIS_LOOP_194_3	10	10
kmeansKernel	kmeansKernel	dupStrm_64_8_7_s	70016	70016
kmeansKernel	kmeansKernel	stream_n_to_one_read_64_8_Pipeline_VITIS_LOOP_111_2	undef	undef
kmeansKernel	kmeansKernel	stream_n_to_one_read_64_8_Pipeline_VITIS_LOOP_127_4	12	12
kmeansKernel	kmeansKernel	stream_n_to_one_read_64_8_s	undef	undef
kmeansKernel	kmeansKernel	stream_n_to_one_collect_64_64_8_Pipeline_VITIS_LOOP_174_1	undef	undef
kmeansKernel	kmeansKernel	stream_n_to_one_collect_64_64_8_s	undef	undef
kmeansKernel	kmeansKernel	stream_n_to_one_distribute_64_64_8_Pipeline_VITIS_LOOP_233_1	undef	undef
kmeansKernel	kmeansKernel	stream_n_to_one_distribute_64_64_8_Pipeline_VITIS_LOOP_252_2	10	10
kmeansKernel	kmeansKernel	stream_n_to_one_distribute_64_64_8_s	undef	undef
kmeansKernel	kmeansKernel	stream_n_to_one_round_robin_64_64_8_s	undef	undef
kmeansKernel	kmeansKernel	streamNToOne_64_64_8_s	undef	undef
kmeansKernel	kmeansKernel	split_64_1_55_Pipeline_VITIS_LOOP_163_1	550002	550002
kmeansKernel	kmeansKernel	split_64_1_55_s	550004	550004
kmeansKernel	kmeansKernel	storeCenters2Array_double_55_100_715_8_1_Pipeline_VITIS_LOOP_191_1	5725	5725
kmeansKernel	kmeansKernel	storeCenters2Array_double_55_100_715_8_1_s	5727	5727
kmeansKernel	kmeansKernel	scanAndStoreCenters_double_55_100_715_8_1_s	undef	undef
kmeansKernel	kmeansKernel	isConverged_double_55_100_715_8_1_Pipeline_LOOP1_LOOP2_LOOP3	45787	45787

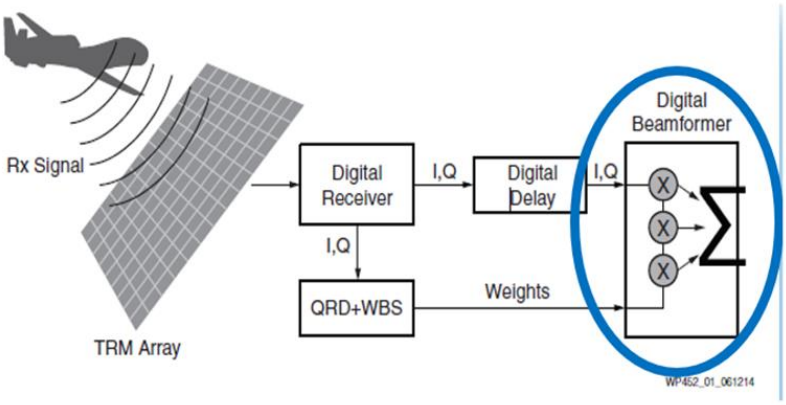
iii. Hardware Target: Co-simulation

The co-simulation is failed in target hardware stage.

```
k=0   c=(-3.00427(-7.0235),4.24077(3.841),-5.05518(-4.5762),10.6768(4.6592),-10.3498(9.1547),6.9251(-13.1232),-3.6063(-
2.6128),-0.0162(-0.1984),1.29855(1.0497),8.06235(20.0001),-22.4459(-21.595),15.8065(5.7262),1.3865(18.1881),11.6521(43.
9962),-15.4567(-17.5136),30.9415(23.2845),-15.5363(-11.8545),0.4058(-25.3256),-3.75117(15.7375),1.77515(-6.4149),2.4752
5(-7.9105),10.3745(-12.1266),3.11717(7.4307),-5.3033(-10.7888),5.59583(33.3725),8.83287(34.1628),23.0399(25.3733),-0.64
4825(9.7484),39.3498(-24.9639),-9.86333(13.6049),-21.7246(-37.2646),8.64813(-2.5249),-5.21075(4.8771),-4.39268(4.1876),
10.6659(-14.8134),7.88(9.2929),7.83415(7.7633),-22.2424(-18.275),-6.05463(3.4093),-27.0167(-19.8583),-54.3251(-39.1586)
,10.5566(24.3457),-3.22487(23.0134),1.13218(3.8273),1.62848(-12.4945),126.677(212.302),153.302(223.106),156.99(165.13),
146.431(130.356),270.265(71.6991),389.778(287.204),314.325(267.011),319.071(267.811),319.024(270.069),199.558(264.831))
k=1   c=(44.0979(-2.16455),0.706133(-2.91375),-3.89773(5.46165),9.79177(7.50635),1.4701(16.0444),-2.30837(-8.1331),15.4
822(24.2404),-13.3479(-7.7601),11.4301(17.5475),-15.9269(9.9706),-2.08403(-12.4398),-16.8002(-19.3473),18.5986(26.3056)
,9.99117(-23.5757),2.21387(-14.5632),-8.72753(10.0969),-1.7795(6.7965),4.1268(-18.9964),11.9839(17.2862),-9.91407(-9.34
785),-6.78307(-2.7512),4.9561(13.0638),0.286667(-7.08975),-0.356067(9.10465),-9.3345(-15.8399),-15.9061(6.3717),-1.2281
7(-13.213),0.766433(32.9348),11.8925(18.1461),4.4379(21.887),3.77043(-26.5373),13.2098(11.8797),16.863(18.008),-0.68936
7(-5.25065),-12.98(-18.505),-2.53823(18.2694),-0.1467(-9.4421),16.1681(19.0502),2.33327(-38.331),18.8722(65.6321),-40.6
368(-34.8815),-4.79463(-5.64815),-13.0863(1.2364),7.76983(-1.4833),-3.98907(3.26695),90.1893(50.0556),87.4569(45.4136),
143.691(117.891),137.692(107.081),175.333(135.804),282.121(371.375),334.198(400.978),359.779(399.886),355.9(401.687),36
9.843(403.928))
k=2   c=(5.2124(5.2124),14.2644(14.2644),-28.3224(-28.3224),21.1945(21.1945),-21.3872(-21.3872),5.9805(5.9805),-23.1323
(-23.1323),-27.1067(-27.1067),25.9398(25.9398),8.0974(8.0974),17.5096(17.5096),-0.124(-0.124),45.1755(45.1755),-24.3029
(-24.3029),13.5472(13.5472),17.2812(17.2812),-3.9665(-3.9665),-30.6986(-30.6986),-2.1054(-2.1054),4.9772(4.9772),-1.002
1(-1.0021),28.9423(28.9423),3.8976(3.8976),-19.392(-19.392),-18.039(-18.039),-36.2282(-36.2282),33.4151(33.4151),-33.15
91(-33.1591),-8.3008(-8.3008),24.1687(24.1687),-17.461(-17.461),-32.4297(-32.4297),-17.69(-17.69),16.9823(16.9823),6.63
9(6.639),20.7256(20.7256),12.7038(12.7038),-25.3189(-25.3189),-30.1496(-30.1496),29.4068(29.4068),-22.3589(-22.3589),-1
3.5126(-13.5126),55.6739(55.6739),11.8744(11.8744),-23.3291(-23.3291),204.081(204.081),212.225(212.225),165.177(165.177
),61.5124(61.5124),224.782(224.782),318.226(318.226),483.163(483.163),476.313(476.313),482.153(482.153),489.288(489.288
))
```

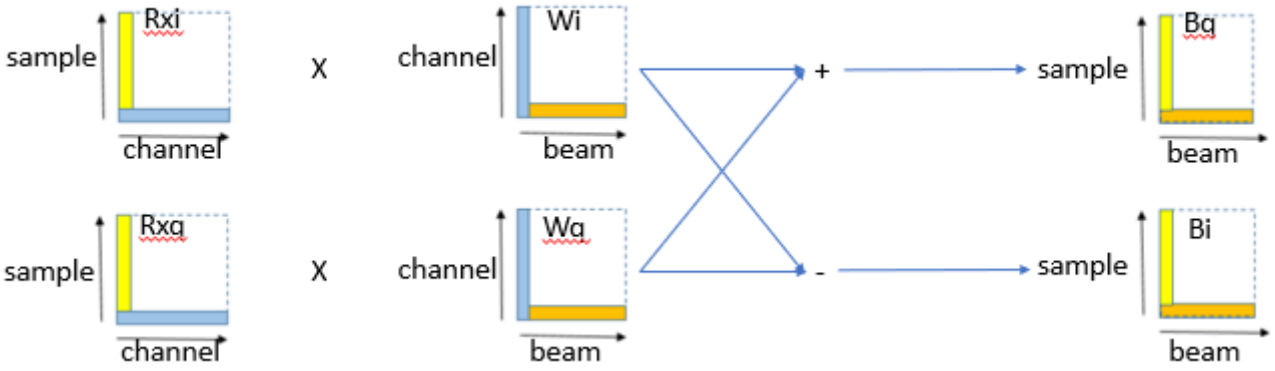
VI. Beamforming Acceleration – Introduction

Beamforming is a signal processing technique used in sensor arrays for directional signal transmission or reception. This is achieved by combining elements in an antenna array in such a way that signals at particular angles experience constructive interference while others experience destructive interference. Beamforming can be used for radio or sound waves. It has found numerous applications in radar, sonar, seismology, wireless communications, radio astronomy, acoustics and biomedicine. We will focus on the circled in the diagram below.



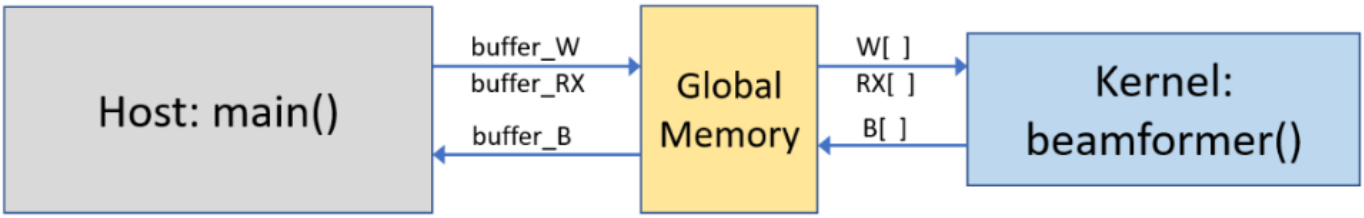
VII. Beamforming Acceleration – Function Description

The complex sensor data (RX_I and RX_Q) is delivered to the Beamformer module as a 2D array [SAMPLES][CHANNELS]. Similarly, the adaptive weights (W_I & W_Q) are also stored as a 2D array [BEAMS][CHANNELS]. Output Beam data (beams_i & beams_q) is computed for each sample [BEAMS][Sample]. The matrix calculation process is shown below. The number of SAMPLES represents the Pulse Repetition Interval (PRI). We need to achieve a PRI < 200 uS for WP452 application .



VIII. Beamforming Acceleration – System Description

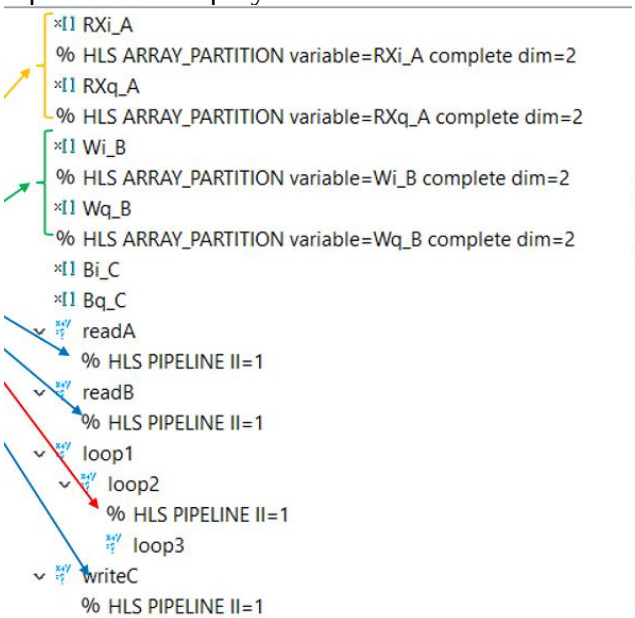
The overall application structure is represented in this block diagram with all the Beamformer calculations done by the Kernel C++ code in 'beamformer()'.



The dataflow between the Host and Kernel. The API commands in the HOST code will be spotlighted that control the data movement for steps 4-7 in the Execution model shown here.

IX. Beamforming Acceleration – HLS directive

In order to achieve higher performance, we tried lots of different pragmas. The array_partition directive is the most important because it can remove memory bottlenecks. We have attempt to use pipeline for read and write, also attempt to use unroll for matrix calulation. In the end, we use array_partition, and pipeline II=1 for three kind of loops. The directive we use and the latency&utilization is shown below. It is OK that utilization exceed 100% because we use pynq-z2 to test, but we will use x50 platform to deploy our hardware.



Latency (cycles)		Latency (absolute)		Interval (cycles)		
min	max	min	max	min	max	Type
55088	55088	1.653 ms	1.653 ms	55089	55089	no

Summary

Name	BRAM_18K	DSP	FF	LUT	URAM
DSP	-	-	-	-	-
Expression	-	-	0	6	-
FIFO	-	-	-	-	-
Instance	8	192	7197	7056	-
Memory	288	-	0	0	0
Multiplexer	-	-	-	2154	-
Register	-	-	404	-	-
Total	296	192	7601	9216	0
Available	280	220	106400	53200	0
Utilization (%)	105	87	7	17	0

X. Beamforming Acceleration – Results & Discussion

1. Hardware Target: Resources

Name	LUT	LUTAsMem	REG	BRAM	URAM	DSP
Platform	11.54%	2.15%	7.10%	13.24%	0.00%	0.07%
User Budget	100.00%	100.00%	100.00%	100.00%	100.00%	100.00%
Used Resources	1.34%	0.22%	1.37%	13.64%	0.00%	3.23%
Unused Resources	98.66%	99.78%	98.63%	86.36%	100.00%	96.77%
beamformer (1)	1.34%	0.22%	1.37%	13.64%	0.00%	3.23%
beamformer_1	1.34%	0.22%	1.37%	13.64%	0.00%	3.23%

2. Hardware Target: Latency
   Latency = 80073 clocks @ 300Mhz
   Latency absolute time = 267 uS

Timing Information (MHz)

Compute Unit	Kernel Name	Module Name	Target Frequency	Estimated Frequency
beamformer_1	beamformer	beamformer_Pipeline_readA	300.300293	411.015198
beamformer_1	beamformer	beamformer_Pipeline_readB	300.300293	411.015198
beamformer_1	beamformer	beamformer_Pipeline_loop1_loop2	300.300293	426.985474
beamformer_1	beamformer	beamformer_Pipeline_writeC	300.300293	411.015198
beamformer_1	beamformer	beamformer	300.300293	411.015198

Latency Information

Compute Unit	Kernel Name	Module Name	Start Interval	Best (cycles)	Avg (cycles)	Worst (cycles)	Best (absolute)	Avg (absolute)	Worst (absolute)
beamformer_1	beamformer	beamformer_Pipeline_readA	80073	80073	80073	80073	0.267 ms	0.267 ms	0.267 ms
beamformer_1	beamformer	beamformer_Pipeline_readB	169	169	169	169	0.563 us	0.563 us	0.563 us
beamformer_1	beamformer	beamformer_Pipeline_loop1_loop2	7511	7511	7511	7511	25.034 us	25.034 us	25.034 us
beamformer_1	beamformer	beamformer_Pipeline_writeC	15072	15072	15072	15072	50.235 us	50.235 us	50.235 us
beamformer_1	beamformer	beamformer	102833	102832	102832	102832	0.343 ms	0.343 ms	0.343 ms

XI. Work Distribution

1. Processed data preparation: 馬婕芸
2. C sources preparation (Spike Sorting): 馬婕芸
3. C sources preparation (Beamforming): 呂易縉、賴聖耘
4. Directives preparation: 馬婕芸、呂易縉、賴聖耘
5. HLS simulation: 馬婕芸、呂易縉、賴聖耘
6. Report writing: 馬婕芸、呂易縉、賴聖耘
7. Presentation: 馬婕芸、呂易縉、賴聖耘

XII. Reference

1. Spike Detection: https://ieeexplore.ieee.org/document/6070974
2. Spike Alignment: https://ieeexplore.ieee.org/document/1608524/
3. Feature Extraction: https://www.sciencedirect.com/science/article/pii/S0165027000002508
4. K-means Clustering: https://xilinx.github.io/Vitis_Accel_Examples/2020.2/html/kmeans.html
5. KiloSort: https://github.com/cortex-lab/KiloSort
6. Beamforming: https://www.xilinx.com/developer/articles/beamforming-acceleration.html

XIII. GitHub: https://github.com/jieyunma/AAHLS_final.git