Real Time Automatic Modulation to Vitis

Motivation

The code aims at automatic modulation classification for radio signals leveraging the RF capabilities offered by Xilinx ZCU111 RFSoC platform. The original code was created using python 3.6 and tensorflow 1.14 using the Vgg10 architecture. The network was trained using the radio modulation dataset created by O'Shea et. al available at deepsig.io.

The C code (trained neural network layers) was generated using a python script and implemented the quantized, low precision CNN components to mimic the precision of Verilog.

The top function compute_network() takes an input image signal of 1024 I/Q samples of the form [I0,Q0,I1,Q1,.., I1023, Q1023]. Output is a 4x1 vector of integers. The model is trained to detect 4 different modulation methods. We apply the softmax function to the output vector to choose between 1 of 4 modulation methods: OOK (0),4ASK (1), 8ASK (2), BPSK (3). The softmax is the index where the maximum value occurs in an array.

We check the accuracy of the output against a validation set for different SNRs, and record the accuracy (100*correct choices / incorrect choices) in output.txt.

Testing With Python or HLS

Code under test/

We improved upon the given code at https://github.com/da-steve101/radio_modulation. Under https://github.com/da-steve101/radio_modulation/tree/master/c, their testing with python does not work with the deepsig.io dataset (they parsed it in their own way and did not link to the file) and is not documented.

DEPENDENCIES:

- tensorflow
- python3
- debian machine for bash scripts

SETUP (for testing): In linux, download tar.gz from here: deepsig.io to code/

```
cp tw_vgg10_hardware.c tw_vgg10.c
bash get_txt_files.sh
export MODEL_DIR=../models/vgg_twn_rfsoc_50k_64_d128
rm -rf build
python3 setup.py build
export PYTHONPATH=build/lib.linux-x86_64-3.6
python3 test_vggpy.py
```

Or run:

```
cp tw_vgg10_hardware.c tw_vgg10.c; bash run_all.sh
```

For each snr in the validation set, accuracy is reported output.txt

For the python testbench, the output should be:

```
        accr[16] = 25.714285714285715

        ...
```

```
accr[-14] =
23.529411764705884

accr[28] = 17.857142857142858
```

Where, for example, at SNR=-14 the accuracy is 23.529.

In Vitis HLS and in hardware emulation on Vitis V++ compiler, we feed a single 2048 input array. The output should *always* display:

FAILED: (0,0) where **-186**!=210

FAILED: (0,1) where **199**!=144

FAILED: (0,2) where **34**!=-144

FAILED: (0,3) where **22**!=46

We used a legacy testbench from here https://github.com/da-steve101/radio_modulation/blob/master/c/tw_vgg10.c, and use wrong values to compare against the correct values (LHS, in bold above). In this way, we can make sure that the first 4 values are always the same. We also need to make sure that the next values are 0. Note that it would choose modulation classification #1 in this case. There are many output vectors that would provide a false negative, and this is one arbitrary way we chose to test our implementation.

Initial Implementation with Vitis HLS

Note: Vitis HLS is found in $(Vitis_install_path)/2019.2/bin/vitis_hls. Its newer version of vivado HLS, and allows us to create .xo files after synthesis that can be used on the Vitis Unified Software Platform.

Baseline (In Vitis HLS):

Code under baseline/
The baseline folder contains the following files:

- src folder:
  - Top file: tw_vgg10.c (and header file tw_vgg10.h)
  - Helper files: all CNN layer (.c and .h) such as conv1.c, dense1.c
  - Testbench file: pyvgg.c
  - Script: script.tcl
- Report folder: contains the synthesis .rpt and .xml

Target Board: ZCU111 (part: xczu28dr-ffvg1517-2-e)

The generated C code from python implementation was not synthesizable. The use of pointer structures was throwing a lot of errors on synthesizing the code. We transformed the code to get a synthesizable baseline implementation in Vitis HLS.

Performance and resource utilization for our baseline implementation are given below.

The baseline implementation has a frequency of 132.1 Hz. As seen from resource utilization, the LUTs exceeds the resource utilization by almost 100%. This is because of the adders/subtractors in the CNN layers.

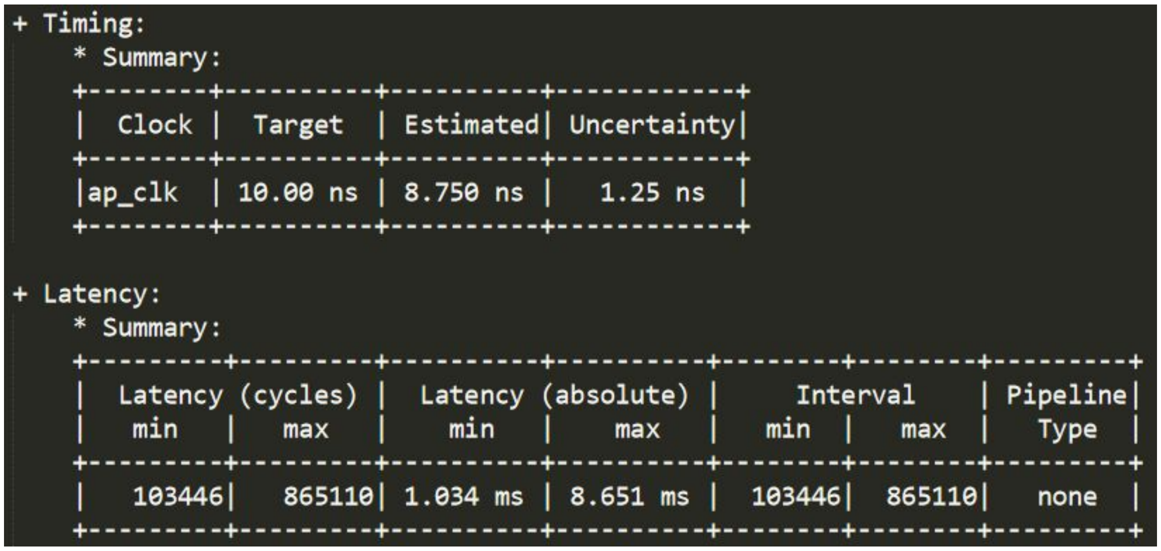

```
+ Timing:
    * Summary:
    +--------+---------+---------+-----------+
    |  Clock |  Target | Estimated| Uncertainty|
    +--------+---------+---------+-----------+
    |ap_clk  | 10.00 ns | 8.750 ns |   1.25 ns  |
    +--------+---------+---------+-----------+

+ Latency:
    * Summary:
    +---------+---------+----------+----------+--------+--------+---------+
    |  Latency (cycles) |  Latency (absolute) |     Interval    | Pipeline|
    |   min   |   max   |    min   |    max   |   min  |   max  |   Type  |
    +---------+---------+----------+----------+--------+--------+---------+
    |   103446|   865110| 1.034 ms | 8.651 ms | 103446|  865110|   none  |
    +---------+---------+----------+----------+--------+--------+---------+
```

Fig- Performance of Baseline implementation

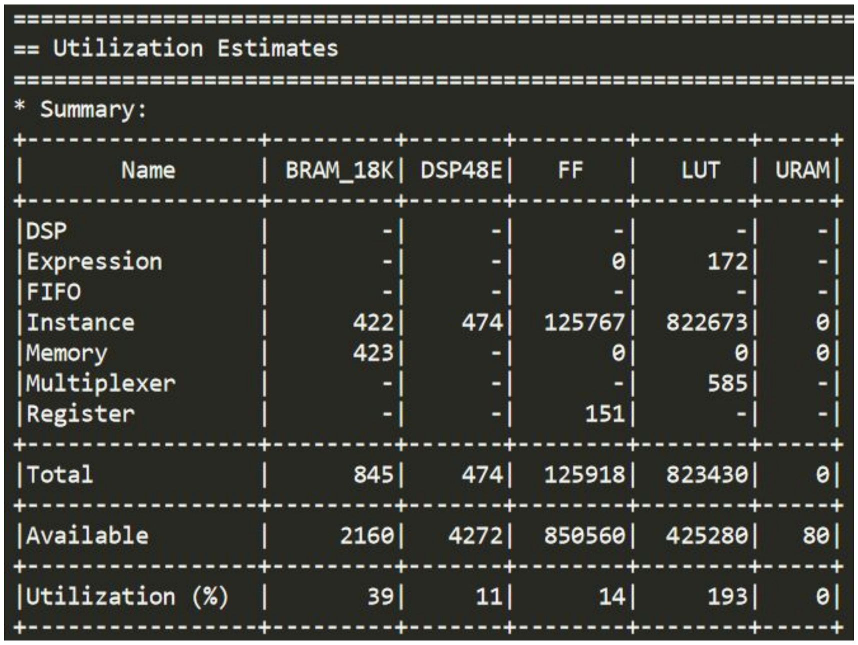

```
================================================================
== Utilization Estimates
================================================================
* Summary:
+-----------------+---------+-------+--------+--------+-----+
|       Name      | BRAM_18K| DSP48E|   FF   |   LUT  | URAM|
+-----------------+---------+-------+--------+--------+-----+
|DSP              |        -|      -|       -|       -|    -|
|Expression       |        -|      -|       0|     172|    -|
|FIFO             |        -|      -|       -|       -|    -|
|Instance         |      422|    474|  125767|  822673|    0|
|Memory           |      423|      -|       0|       0|    0|
|Multiplexer      |        -|      -|       -|     585|    -|
|Register         |        -|      -|     151|       -|    -|
+-----------------+---------+-------+--------+--------+-----+
|Total            |      845|    474|  125918|  823430|    0|
+-----------------+---------+-------+--------+--------+-----+
|Available        |     2160|   4272|  850560|  425280|   80|
+-----------------+---------+-------+--------+--------+-----+
|Utilization (%)  |       39|     11|      14|     193|    0|
+-----------------+---------+-------+--------+--------+-----+
```

Fig- Resource Utilization of Baseline implementation

Optimizing the design in Vitis HLS

Code under optimized/
The optimized folder contains the following files:

- src folder:
    - Top file: tw_vgg10.c (and header file tw_vgg10.h)
    - Helper files: all CNN layer (.c and .h) such as conv1.c, dense1.c
    - Testbench file: pyvgg.c
    - Script: script.tcl
- Report folder: contains the synthesis .rpt and .xml

Target Board: ZCU111 (part: xczu28dr-ffvg1517-2-e)

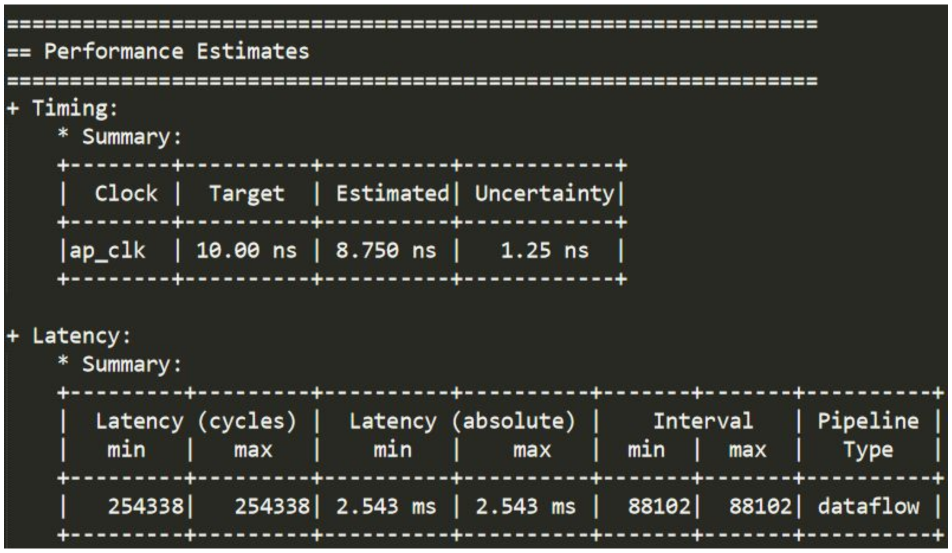
The following optimizations were used to improve the performance and resource utilization of our baseline design:

- Convolution functions and loops and array structure in file tw_vgg10.c was simplified
- Dataflow implementation along with array partitioning (block factor=2) was performed for the design
- LUT Add/sub units were changed to DSP48E Add/Sub units in the CNN layers (conv2.c, conv3.c etc) to decrease our LUT usage
- The loops were pipelined and unrolled to increase throughput of our design

Performance and resource utilization for the optimized implementation are given below.

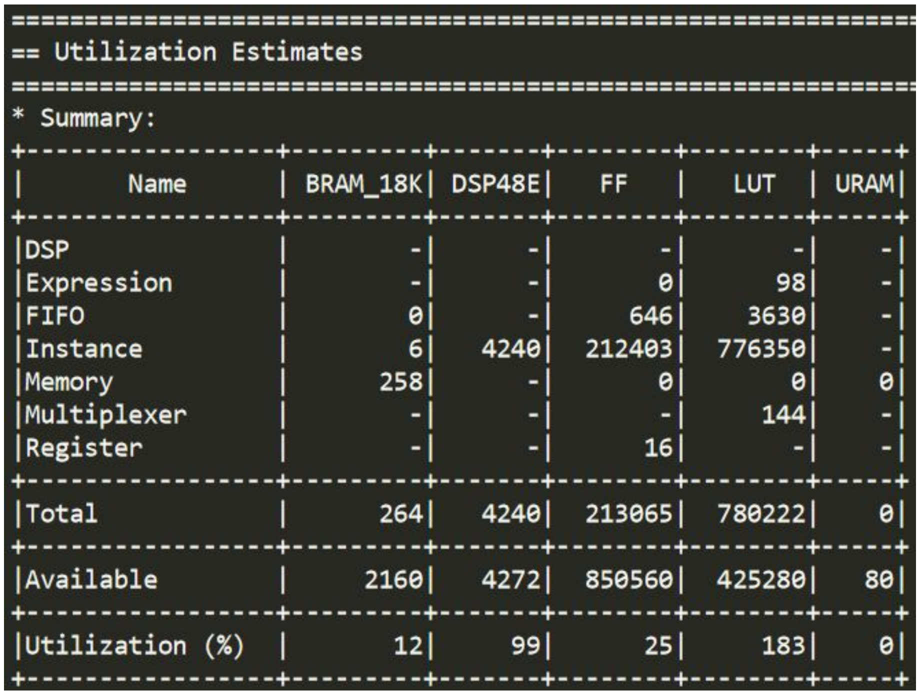
The optimized implementation has a frequency of 450 Hz which is almost 4x that baseline.

Our resource utilization of LUTs decreased significantly from baseline. However, it still exceeds the resource utilization by almost 83%. Even though resource allocation pragmas were used to shift some of the LUT adders/subtractors of the CNN layer files (conv2.c, conv3.c, dense1.c etc) to DSP48E adders/subtractors, the adders/subtractors of the vgg10 architecture could not fit the board using just the Vitis HLS pragmas.



```
================================================================
== Performance Estimates
================================================================
+ Timing:
    * Summary:
    +--------+---------+----------+------------+
    |  Clock |  Target | Estimated| Uncertainty|
    +--------+---------+----------+------------+
    |ap_clk  | 10.00 ns | 8.750 ns |   1.25 ns  |
    +--------+---------+----------+------------+

+ Latency:
    * Summary:
    +---------+---------+----------+----------+-------+-------+----------+
    |  Latency (cycles) |  Latency (absolute) |    Interval   | Pipeline |
    |   min   |   max   |    min   |    max   |  min  |  max  |   Type   |
    +---------+---------+----------+----------+-------+-------+----------+
    |   254338|   254338| 2.543 ms | 2.543 ms |  88102|  88102| dataflow |
    +---------+---------+----------+----------+-------+-------+----------+
```

Fig- Performance of Optimized implementation



```
================================================================
== Utilization Estimates
================================================================
* Summary:
+-----------------+---------+-------+--------+--------+-----+
|       Name      | BRAM_18K| DSP48E|   FF   |   LUT  | URAM|
+-----------------+---------+-------+--------+--------+-----+
|DSP              |        -|      -|       -|       -|    -|
|Expression       |        -|      -|       0|      98|    -|
|FIFO             |        0|      -|     646|    3630|    -|
|Instance         |        6|   4240|  212403|  776350|    -|
|Memory           |      258|      -|       0|       0|    0|
|Multiplexer      |        -|      -|       -|     144|    -|
|Register         |        -|      -|      16|       -|    -|
+-----------------+---------+-------+--------+--------+-----+
|Total            |      264|   4240|  213065|  780222|    0|
+-----------------+---------+-------+--------+--------+-----+
|Available        |     2160|   4272|  850560|  425280|   80|
+-----------------+---------+-------+--------+--------+-----+
|Utilization (%)  |       12|     99|      25|     183|    0|
+-----------------+---------+-------+--------+--------+-----+
```

Fig- Resource utilization of Optimized implementation

Running In Vitis

Code under vitis_emulation/

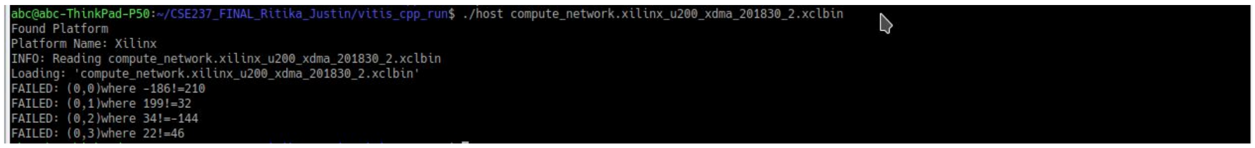
Now that we have optimized code, we run software emulation to check functionality, then hardware emulation in place of the physical chip. Hardware emulation defined by the Vitis toolset means that it emulates the target device. One of the main benefits of Vitis is that after hardware emulation, that is a high degree of confidence that it will run the same on chip due to the portability of the output file: xclbin. The target devices all run a linux cpu layer on top of an fpga, so portability isn't much of a problem.

The target device we use is **u200_xdma_201830_2** in place of ZCU111, as the Vitis does not have support for ZCU111 as of yet**.** See setup.pdf for a walkthrough of how to install vitis and how to install the target device platform on ubuntu 18.04 if you are interested in running on your own machine, or want to expand on this project.

Software and Hardware emulation outputs

The purpose of these outputs is to make sure functionality is correct. The outputs are the same as the vitis hls testbench output. There are also interesting logs showing that axis are being used in place of procedural code.

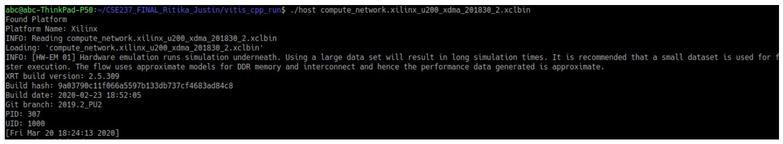
Software Emulation (few minutes run time)



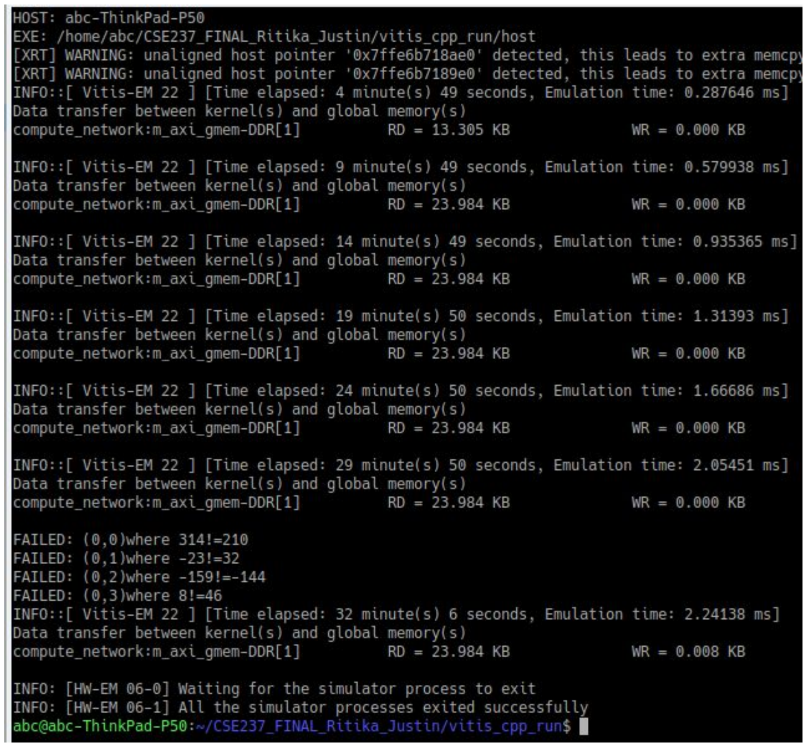
```
abc@abc-ThinkPad-P50:~/CSE237_FINAL_Ritika_Justin/vitis_cpp_run$ ./host compute_network.xilinx_u200_xdma_201830_2.xclbin
Found Platform
Platform Name: Xilinx
INFO: Reading compute_network.xilinx_u200_xdma_201830_2.xclbin
Loading: 'compute_network.xilinx_u200_xdma_201830_2.xclbin'
FAILED: (0,0)where -186!=210
FAILED: (0,1)where 199!=32
FAILED: (0,2)where 34!=-144
FAILED: (0,3)where 22!=46
```

Hardware Emulation is most likely to work if software emulation works.

Hardware Emulation (2hour+ run time)



```
abc@abc-ThinkPad-P50:~/CSE237_FINAL_Ritika_Justin/vitis_cpp_run$ ./host compute_network.xilinx_u200_xdma_201830_2.xclbin
Found Platform
Platform Name: Xilinx
INFO: Reading compute_network.xilinx_u200_xdma_201830_2.xclbin
Loading: 'compute_network.xilinx_u200_xdma_201830_2.xclbin'
INFO: [HW-EM 01] Hardware emulation runs simulation underneath. Using a large data set will result in long simulation times. It is recommended that a small dataset is used for f
ster execution. The flow uses approximate models for DDR memory and interconnect and hence the performance data generated is approximate.
XRT build version: 2.5.309
Build hash: 9a03790c11f066a5597b133db737cf4683ad84c8
Build date: 2020-02-23 18:52:05
Git branch: 2019.2_PU2
PID: 307
UID: 1000
[Fri Mar 20 18:24:13 2020]
```



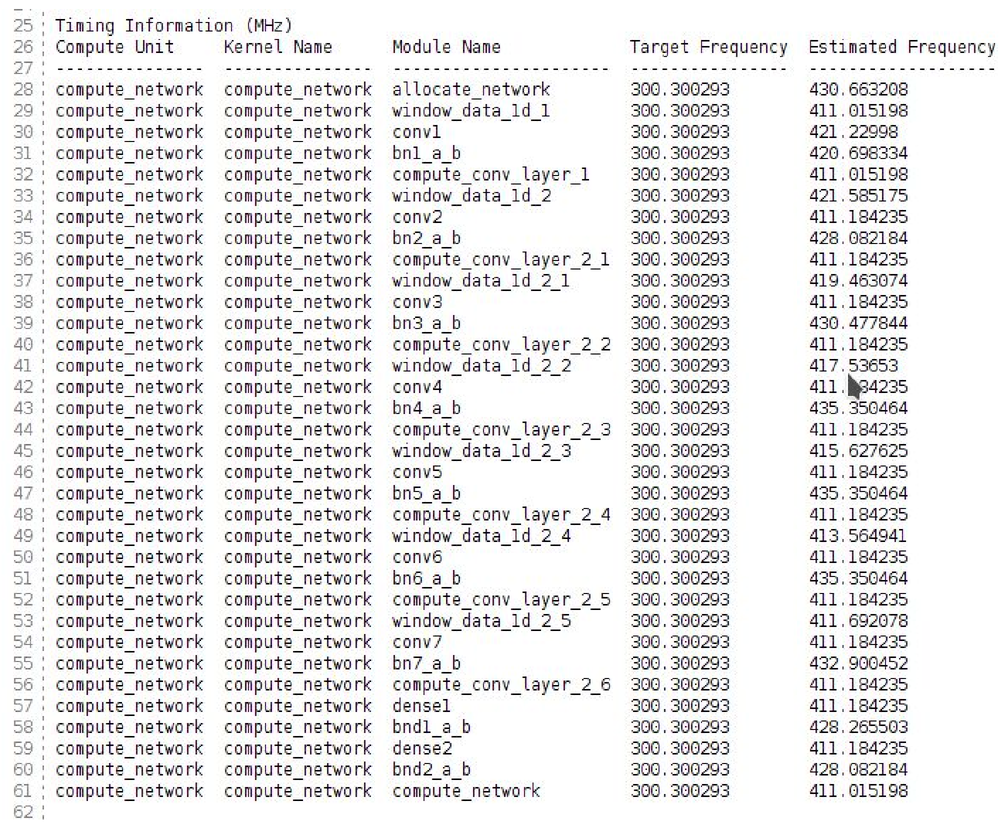
```
HOST: abc-ThinkPad-P50
EXE: /home/abc/CSE237_FINAL_Ritika_Justin/vitis_cpp_run/host
[XRT] WARNING: unaligned host pointer '0x7ffe6b718ae0' detected, this leads to extra memcpy
[XRT] WARNING: unaligned host pointer '0x7ffe6b7189e0' detected, this leads to extra memcpy
INFO::[ Vitis-EM 22 ] [Time elapsed: 4 minute(s) 49 seconds, Emulation time: 0.287646 ms]
Data transfer between kernel(s) and global memory(s)
compute_network:m_axi_gmem-DDR[1]          RD = 13.305 KB              WR = 0.000 KB

INFO::[ Vitis-EM 22 ] [Time elapsed: 9 minute(s) 49 seconds, Emulation time: 0.579938 ms]
Data transfer between kernel(s) and global memory(s)
compute_network:m_axi_gmem-DDR[1]          RD = 23.984 KB              WR = 0.000 KB

INFO::[ Vitis-EM 22 ] [Time elapsed: 14 minute(s) 49 seconds, Emulation time: 0.935365 ms]
Data transfer between kernel(s) and global memory(s)
compute_network:m_axi_gmem-DDR[1]          RD = 23.984 KB              WR = 0.000 KB

INFO::[ Vitis-EM 22 ] [Time elapsed: 19 minute(s) 50 seconds, Emulation time: 1.31393 ms]
Data transfer between kernel(s) and global memory(s)
compute_network:m_axi_gmem-DDR[1]          RD = 23.984 KB              WR = 0.000 KB

INFO::[ Vitis-EM 22 ] [Time elapsed: 24 minute(s) 50 seconds, Emulation time: 1.66686 ms]
Data transfer between kernel(s) and global memory(s)
compute_network:m_axi_gmem-DDR[1]          RD = 23.984 KB              WR = 0.000 KB

INFO::[ Vitis-EM 22 ] [Time elapsed: 29 minute(s) 50 seconds, Emulation time: 2.05451 ms]
Data transfer between kernel(s) and global memory(s)
compute_network:m_axi_gmem-DDR[1]          RD = 23.984 KB              WR = 0.000 KB

FAILED: (0,0)where 314!=210
FAILED: (0,1)where -23!=32
FAILED: (0,2)where -159!=-144
FAILED: (0,3)where 8!=46
INFO::[ Vitis-EM 22 ] [Time elapsed: 32 minute(s) 6 seconds, Emulation time: 2.24138 ms]
Data transfer between kernel(s) and global memory(s)
compute_network:m_axi_gmem-DDR[1]          RD = 23.984 KB              WR = 0.008 KB

INFO: [HW-EM 06-0] Waiting for the simulator process to exit
INFO: [HW-EM 06-1] All the simulator processes exited successfully
abc@abc-ThinkPad-P50:~/CSE237_FINAL_Ritika_Justin/vitis_cpp_run$
```

Final Report files

After hardware emulation is run, final estimates are taken before being run on the board. The compute_network.xilinx_u200_xdma_201830_2.xclbin file in vitis_emulation/ can be run on the sd card of the xilinx_u200_xdma_201830_2 board.

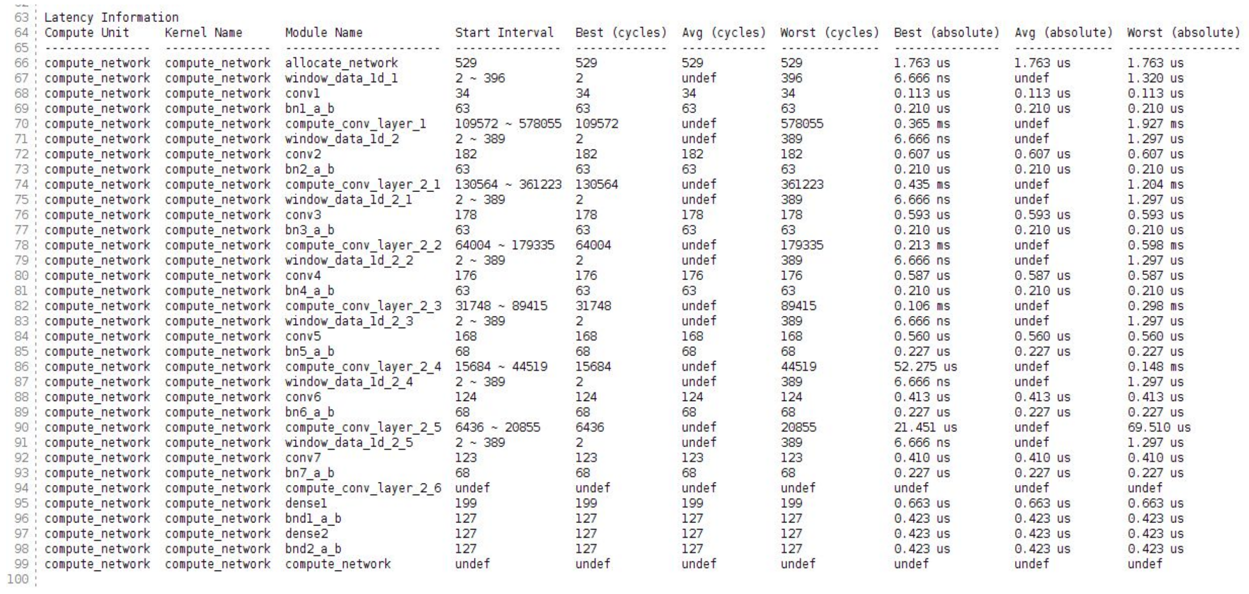
The figures on the next few pages show the final estimates given by Vitis.

Figure 1



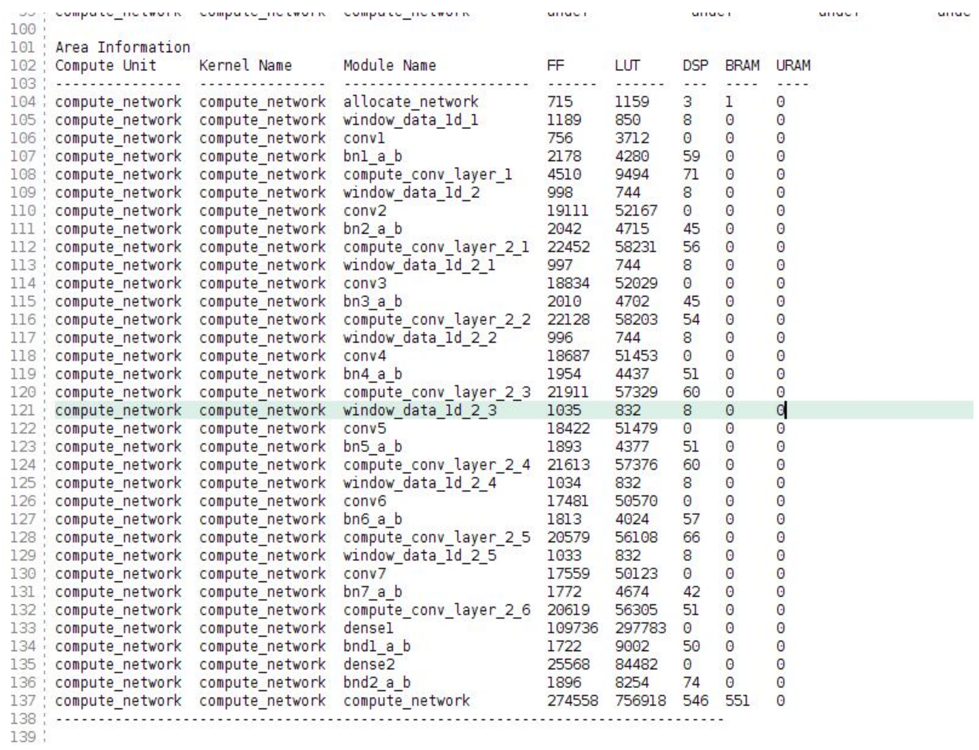
```
Timing Information (MHz)
Compute Unit     Kernel Name      Module Name            Target Frequency  Estimated Frequency
---------------  ---------------  ---------------------  ----------------  -------------------
compute_network  compute_network  allocate_network       300.300293        430.663208
compute_network  compute_network  window_data_ld_1       300.300293        411.015198
compute_network  compute_network  conv1                  300.300293        421.22998
compute_network  compute_network  bn1_a_b                300.300293        420.698334
compute_network  compute_network  compute_conv_layer_1   300.300293        411.015198
compute_network  compute_network  window_data_ld_2       300.300293        421.585175
compute_network  compute_network  conv2                  300.300293        411.184235
compute_network  compute_network  bn2_a_b                300.300293        428.082184
compute_network  compute_network  compute_conv_layer_2_1 300.300293        411.184235
compute_network  compute_network  window_data_ld_2_1     300.300293        419.463074
compute_network  compute_network  conv3                  300.300293        411.184235
compute_network  compute_network  bn3_a_b                300.300293        430.477844
compute_network  compute_network  compute_conv_layer_2_2 300.300293        411.184235
compute_network  compute_network  window_data_ld_2_2     300.300293        417.53653
compute_network  compute_network  conv4                  300.300293        411.184235
compute_network  compute_network  bn4_a_b                300.300293        435.350464
compute_network  compute_network  compute_conv_layer_2_3 300.300293        411.184235
compute_network  compute_network  window_data_ld_2_3     300.300293        415.627625
compute_network  compute_network  conv5                  300.300293        411.184235
compute_network  compute_network  bn5_a_b                300.300293        435.350464
compute_network  compute_network  compute_conv_layer_2_4 300.300293        411.184235
compute_network  compute_network  window_data_ld_2_4     300.300293        413.564941
compute_network  compute_network  conv6                  300.300293        411.184235
compute_network  compute_network  bn6_a_b                300.300293        435.350464
compute_network  compute_network  compute_conv_layer_2_5 300.300293        411.184235
compute_network  compute_network  window_data_ld_2_5     300.300293        411.692078
compute_network  compute_network  conv7                  300.300293        411.184235
compute_network  compute_network  bn7_a_b                300.300293        432.900452
compute_network  compute_network  compute_conv_layer_2_6 300.300293        411.184235
compute_network  compute_network  dense1                 300.300293        411.184235
compute_network  compute_network  bnd1_a_b               300.300293        428.265503
compute_network  compute_network  dense2                 300.300293        411.184235
compute_network  compute_network  bnd2_a_b               300.300293        428.082184
compute_network  compute_network  compute_network        300.300293        411.015198
```

Figure 2



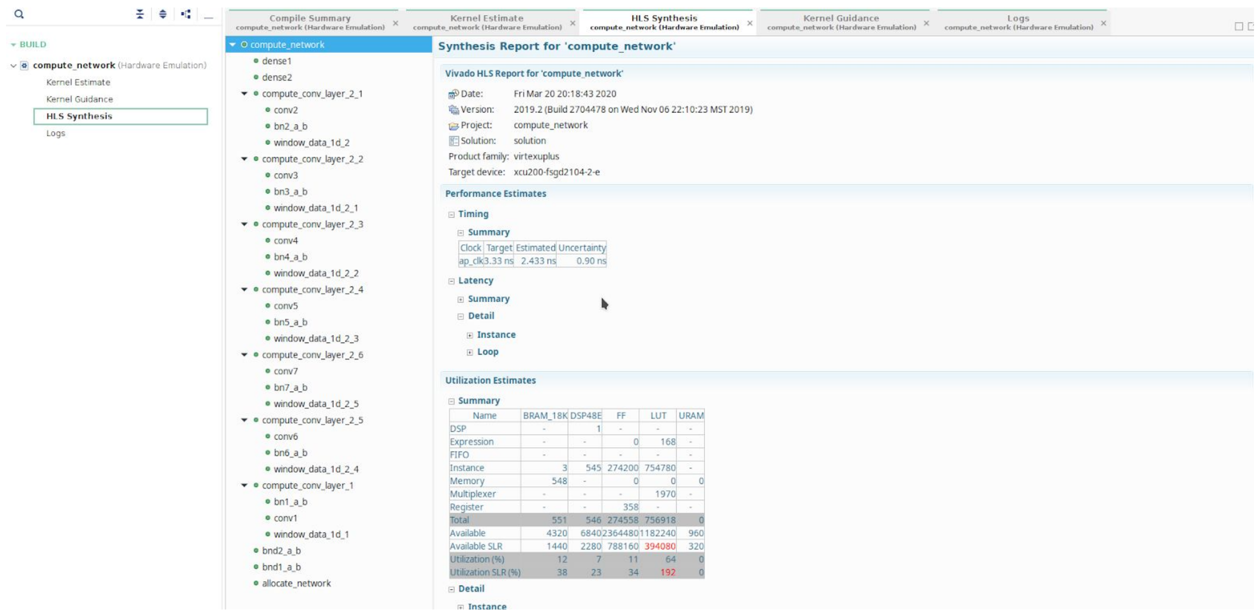
```
Latency Information
Compute Unit     Kernel Name      Module Name            Start Interval   Best (cycles)  Avg (cycles)  Worst (cycles)  Best (absolute)  Avg (absolute)  Worst (absolute)
---------------  ---------------  ---------------------  ---------------  -------------  ------------  --------------  ---------------  --------------  ----------------
compute_network  compute_network  allocate_network       529              529            529           529             1.763 us         1.763 us        1.763 us
compute_network  compute_network  window_data_ld_1       2 ~ 396          2              undef         396             6.666 ns         undef           1.320 us
compute_network  compute_network  conv1                  34               34             34            34              0.113 us         0.113 us        0.113 us
compute_network  compute_network  bn1_a_b                63               63             63            63              0.210 us         0.210 us        0.210 us
compute_network  compute_network  compute_conv_layer_1   109572 ~ 578055  109572         undef         578055          0.365 ms         undef           1.927 ms
compute_network  compute_network  window_data_ld_2       2 ~ 389          2              undef         389             6.666 ns         undef           1.297 us
compute_network  compute_network  conv2                  182              182            182           182             0.607 us         0.607 us        0.607 us
compute_network  compute_network  bn2_a_b                63               63             63            63              0.210 us         0.210 us        0.210 us
compute_network  compute_network  compute_conv_layer_2_1 130564 ~ 361223  130564         undef         361223          0.435 ms         undef           1.204 ms
compute_network  compute_network  window_data_ld_2_1     2 ~ 389          2              undef         389             6.666 ns         undef           1.297 us
compute_network  compute_network  conv3                  178              178            178           178             0.593 us         0.593 us        0.593 us
compute_network  compute_network  bn3_a_b                63               63             63            63              0.210 us         0.210 us        0.210 us
compute_network  compute_network  compute_conv_layer_2_2 64004 ~ 179335   64004          undef         179335          0.213 ms         undef           0.598 ms
compute_network  compute_network  window_data_ld_2_2     2 ~ 389          2              undef         389             6.666 ns         undef           1.297 us
compute_network  compute_network  conv4                  176              176            176           176             0.587 us         0.587 us        0.587 us
compute_network  compute_network  bn4_a_b                63               63             63            63              0.210 us         0.210 us        0.210 us
compute_network  compute_network  compute_conv_layer_2_3 31748 ~ 89415    31748          undef         89415           0.106 ms         undef           0.298 ms
compute_network  compute_network  window_data_ld_2_3     2 ~ 389          2              undef         389             6.666 ns         undef           1.297 us
compute_network  compute_network  conv5                  168              168            168           168             0.560 us         0.560 us        0.560 us
compute_network  compute_network  bn5_a_b                68               68             68            68              0.227 us         0.227 us        0.227 us
compute_network  compute_network  compute_conv_layer_2_4 15684 ~ 44519    15684          undef         44519           52.275 us        undef           0.148 ms
compute_network  compute_network  window_data_ld_2_4     2 ~ 389          2              undef         389             6.666 ns         undef           1.297 us
compute_network  compute_network  conv6                  124              124            124           124             0.413 us         0.413 us        0.413 us
compute_network  compute_network  bn6_a_b                68               68             68            68              0.227 us         0.227 us        0.227 us
compute_network  compute_network  compute_conv_layer_2_5 6436 ~ 20855     6436           undef         20855           21.451 us        undef           69.510 us
compute_network  compute_network  window_data_ld_2_5     2 ~ 389          2              undef         389             6.666 ns         undef           1.297 us
compute_network  compute_network  conv7                  123              123            123           123             0.410 us         0.410 us        0.410 us
compute_network  compute_network  bn7_a_b                68               68             68            68              0.227 us         0.227 us        0.227 us
compute_network  compute_network  compute_conv_layer_2_6 undef            undef          undef         undef           undef            undef           undef
compute_network  compute_network  dense1                 199              199            199           199             0.663 us         0.663 us        0.663 us
compute_network  compute_network  bnd1_a_b               127              127            127           127             0.423 us         0.423 us        0.423 us
compute_network  compute_network  dense2                 127              127            127           127             0.423 us         0.423 us        0.423 us
compute_network  compute_network  bnd2_a_b               127              127            127           127             0.423 us         0.423 us        0.423 us
compute_network  compute_network  compute_network        undef            undef          undef         undef           undef            undef           undef
```

Figure 3



```
Area Information
Compute Unit     Kernel Name      Module Name             FF      LUT     DSP  BRAM  URAM
---------------  ---------------  ----------------------  ------  ------  ---  ----  ----
compute_network  compute_network  allocate_network        715     1159    3    1     0
compute_network  compute_network  window_data_1d_1        1189    850     8    0     0
compute_network  compute_network  conv1                   756     3712    0    0     0
compute_network  compute_network  bn1_a_b                 2178    4280    59   0     0
compute_network  compute_network  compute_conv_layer_1    4510    9494    71   0     0
compute_network  compute_network  window_data_1d_2        998     744     8    0     0
compute_network  compute_network  conv2                   19111   52167   0    0     0
compute_network  compute_network  bn2_a_b                 2042    4715    45   0     0
compute_network  compute_network  compute_conv_layer_2_1  22452   58231   56   0     0
compute_network  compute_network  window_data_1d_2_1      997     744     8    0     0
compute_network  compute_network  conv3                   18834   52029   0    0     0
compute_network  compute_network  bn3_a_b                 2010    4702    45   0     0
compute_network  compute_network  compute_conv_layer_2_2  22128   58203   54   0     0
compute_network  compute_network  window_data_1d_2_2      996     744     8    0     0
compute_network  compute_network  conv4                   18687   51453   0    0     0
compute_network  compute_network  bn4_a_b                 1954    4437    51   0     0
compute_network  compute_network  compute_conv_layer_2_3  21911   57329   60   0     0
compute_network  compute_network  window_data_1d_2_3      1035    832     8    0     0
compute_network  compute_network  conv5                   18422   51479   0    0     0
compute_network  compute_network  bn5_a_b                 1893    4377    51   0     0
compute_network  compute_network  compute_conv_layer_2_4  21613   57376   60   0     0
compute_network  compute_network  window_data_1d_2_4      1034    832     8    0     0
compute_network  compute_network  conv6                   17481   50570   0    0     0
compute_network  compute_network  bn6_a_b                 1813    4024    57   0     0
compute_network  compute_network  compute_conv_layer_2_5  20579   56108   66   0     0
compute_network  compute_network  window_data_1d_2_5      1033    832     8    0     0
compute_network  compute_network  conv7                   17559   50123   0    0     0
compute_network  compute_network  bn7_a_b                 1772    4674    42   0     0
compute_network  compute_network  compute_conv_layer_2_6  20619   56305   51   0     0
compute_network  compute_network  dense1                  109736  297783  0    0     0
compute_network  compute_network  bnd1_a_b                1722    9002    50   0     0
compute_network  compute_network  dense2                  25568   84482   0    0     0
compute_network  compute_network  bnd2_a_b                1896    8254    74   0     0
compute_network  compute_network  compute_network         274558  756918  546  551   0
---------------------------------------------------------------------------------------------
```

Figure 4



Synthesis Report for 'compute_network'

Vivado HLS Report for 'compute_network'

Date:	Fri Mar 20 20:18:43 2020
Version:	2019.2 (Build 2704478 on Wed Nov 06 22:10:23 MST 2019)
Project:	compute_network
Solution:	solution
Product family:	virtexuplus
Target device:	xcu200-fsgd2104-2-e

Performance Estimates

Timing

Summary

Clock	Target	Estimated	Uncertainty
ap_clk	3.33 ns	2.433 ns	0.90 ns

Latency

Summary

Detail

Instance

Loop

Utilization Estimates

Summary

Name	BRAM_18K	DSP48E	FF	LUT	URAM
DSP	-	1	-	-	-
Expression	-	-	0	168	-
FIFO	-	-	-	-	-
Instance	3	545	274200	754780	-
Memory	548	-	0	0	0
Multiplexer	-	-	-	1970	-
Register	-	-	358	-	-
Total	551	546	274558	756918	0
Available	4320	6840	2364480	1182240	960
Available SLR	1440	2280	788160	394080	320
Utilization (%)	12	7	11	64	0
Utilization SLR (%)	38	23	34	192	0

Detail

Instance

If the run is successful, you should get two classes of report files:

- Get report files from hardware emulation (this runs vivado hls or vitis hls).
    - Run vitis_analyzer in the <Vitis_install_location>/2019.2/bin and open the link summaries to get the Estimated Timing Information (Figure 1), Latency Information (Figure 2) and Area Information (Figure 3).
    - Open the Compile Summary to get the estimated latency and area on the board (figure 4).

Figure 4 shows that the design fits for our target device - u200_xdma_201830_2. Although, it estimates that the SLR portion of the board is overutilized. This is just an estimate, and we can't change this without running board specific optimizations (which is counterproductive because we want to eventually migrate to ZCU111) [1].

[1]: https://www.xilinx.com/html_docs/xilinx2019_1/sdaccel_doc/xqd1554752221979.html#phm1523550940155.

Tradeoffs between Vitis and Vivado

Figure A

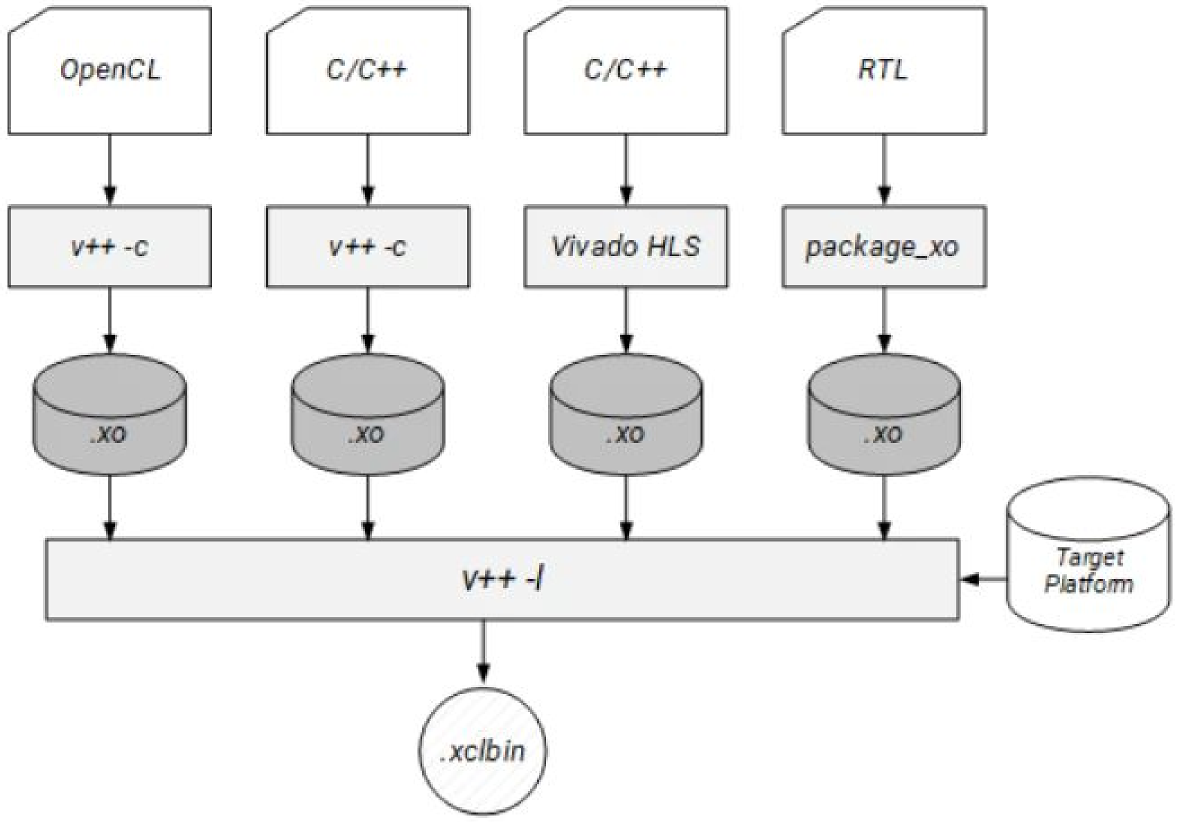


Figure A is from https://www.hackster.io/news/microzed-chronicles-vitis-hls-85a9b1d272e6

Pros of the Vitis Unified Software Platform

- Uses host file and a kernel file
- See diagram. Kernel is "plug-and-play" i.e. it can be linked with many programming languages for the host file.
- Allows for multiple programming languages. C for the kernel. The host can be cpp, c or python.
- See the flow below, it allows us to test in software with axis, before running on hardware. Our design was medium sized, and it took around 2 hours to run the Hardware steps below. If our design was large, this would be an even bigger benefit.
- Platforms that are supported are easy to download (the dpkg files are given for ubuntu users).

Cons of the Vitis Unified Software Platform

- Very little documentation
- Still very buggy. For instance, although it supports C, there is a bug that using a c file extension will not allow vivado hls to export the .xo file. This can only be fixed by changing c files to cpp.
- Only really supports linux. There is a gui for windows, but it is documented even less than the ubuntu version. Had to use ubuntu 18.04 to run the tutorials.
- Accelerated Platforms (i.e. platforms that get the upsides mentioned above) are limited for now.

Future Work (Possible CSE 237D Project)

- Run this code on the ZCU111 board using the PYNQ interface, probably have to do vivado as ZCU111 is not supported officially by vitis yet.
- Modify the axi interface to stream 2048 inputs one at a time
- Optimize in vitis, the possibilities are much more than we were able to go though. Memory buffers are the most obvious route to take.
- Optimize in hls, as we learned in class there we can explore further the pareto curve. We made many sacrifices for the area (this machine learning model was 10 layers of 1000-input look up tables) but this was not enough for ZCU111.
- Suggested by Alireza - Try a different architecture, such as imagenet, that is larger but gives better accuracy.