Codesign Challenge - Machine learning Inference

I. Architecture and design strategy

This design contains an HPS, therefore, the final architecture is quite simple. Figure 1 illustrates a high-level abstraction of the design. The main components of the design are a NIOS core and a hard processing system (HPS). The principle idea behind the acceleration process was to delegate the work to efficient hardware. Since we already have a powerful processing unit present, I tried to use it. Just by applying a NIOS and some essential hardware like floating-point unit custom instructions and DMA, the final cycle counts were 1.6 billion (even with compiler optimization flags set for higher performance). That finally steered the design direction in favor of using HPS as an accelerator.

Now, the design could have been improved by using multiple NIOS cores in parallel to HPS. However, the problem would be to transfer data and synchronize the cores. Also, it will increase the design footprint many times. Considering all these and time constraints, this design uses HPS and simple communication between NIOS and HPS. The following steps discuss the overall system operation.

NIOS:

1) Start the timer.
2) Send the addresses of the image, label, training coefficients to the HPS.
3) Wait for HPS to complete the processing
4) Collect the number of errors in the detection process and calculate the cycle.

HPS:

1) Wait for NIOS to initiate data transfer.
2) A simple state machine keeps track of the address that HPS is receiving and assigns the address to pointers.
3) Once three addresses are received, HPS starts processing the data. Since the HPS is being used as an accelerator, it does not print any image in its terminal. To see if the processing is being done correctly, the *verbose* in \software \hps _v2_neuralnetwork\mnist _infertest.c can be set to 1.
4) When the processing is done signal the NIOS to stop the timer.
5) Send the errors to the NIOS.

The NIOS sends the address of the data through an input-output port which is visible to HPS as well. Therefore, one PIO port is used as a synchronization register and another one is used to transfer the addresses. The synchronization is pretty simple: the NIOS sends a specific pattern in a PIO port (PIO_0 in the design) that HPS reads and recognizes are set/reset request from the NIOS. Once these two processors syncs, the NIOS sends the address of a data through PIO_1. The HPS

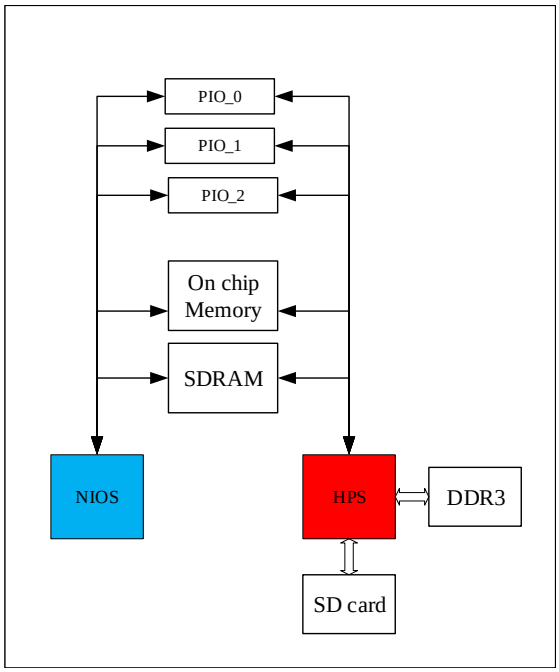

Fig. 1. Top level block diagram of the system

reads the address from that PIO port and stores that to a pointer that would be further used to access the rest of the elements of that location of that data. It is possible for HPS to access the rest of the elements of an array (such as an image) because of the predefined length of that array.

I decided to pass the address only rather than data itself because of the inherent low speed of the NIOS. Instead, the HPS pulls the data from the SDRAM itself. The tensors are constructed at the HPS side once all the data accessed from the SDRAM and stored in the DDR3. By doing this, the design loses cycles to construct the structure because in the reference implementation the structures are constructed before the timer starts. However, in this design, the timer starts as soon as the data transfer starts. This does not add much timing overhead since HPS is a lot faster than NIOS.

A. Software modifications

The NIOS part of the software is in:\software \nios _v2_neuralnetwork
The HPS part of the software is in: \software \hps _v2_neuralnetwork

As mentioned above, the NIOS has no neural network operation. Therefore, all neural net related codes have been commented out from in compiler settings [see Makefile]. The most important files on the NIOS end are the *imagecoef.h* and *trainingcoef.h*. As per the restriction in this project, these files are not present in HPS and the timer starts before the transfer of this data.

Both HPS and NIOS make files contain optimization flags. Please see *Makefile.arm* and *Makefile* in the respective directories.

II. Result

The system consumes 2600400 cycles. Therefore, the speedup is about $65529544770/2600400 = 25200$ times. The cycle consumption in HPS varies depending on the caching and workload of the HPS. However, from repeated measurements, the speedup calculated using the median.

```
----------------------------------------
Please initiate processing by starting HPS
Done processing: Total cycles: 3838271
Errors: 9
Please initiate processing by starting HPS
Done processing: Total cycles: 2597416
Errors: 9
Please initiate processing by starting HPS
Done processing: Total cycles: 2600037
Errors: 9
Please initiate processing by starting HPS
Done processing: Total cycles: 2604505
Errors: 9
Please initiate processing by starting HPS
Done processing: Total cycles: 2610005
Errors: 9
Please initiate processing by starting HPS
Done processing: Total cycles: 2602624
Errors: 9
Please initiate processing by starting HPS
Done processing: Total cycles: 2593002
Errors: 9
Please initiate processing by starting HPS
Done processing: Total cycles: 2593185
Errors: 9
Please initiate processing by starting HPS
Done processing: Total cycles: 2603193
Errors: 9
Please initiate processing by starting HPS
Done processing: Total cycles: 2595967
Errors: 9
Please initiate processing by starting HPS
Done processing: Total cycles: 2600764
Errors: 9
Please initiate processing by starting HPS
Done processing: Total cycles: 2592719
Errors: 9
Please initiate processing by starting HPS
Done processing: Total cycles: 2603328
Errors: 9
Please initiate processing by starting HPS
Done processing: Total cycles: 2588567
Errors: 9
----------------------------------------
```

A. Resource usage

```
+---------------------------------+-----------------------------------------+
; Logic utilization (in ALMs)     ; 3,171 / 32,070 ( 10 % )                  ;
; Total registers                 ; 7124                                      ;
; Total block memory bits         ; 1,116,800 / 4,065,280 ( 27 % )            ;
; Total DSP Blocks                ; 0 / 87 ( 0 % )                           ;
; Total PLLs                      ; 1 / 6 ( 17 % )                           ;
; Total DLLs                      ; 1 / 4 ( 25 % )                           ;
+---------------------------------+-----------------------------------------+
```

III. Conclusion

The design has been accelerated primarily because of using an advanced CPU architecture. Offloading the NIOS from any processing especially helped. Note that, while the HPS processes the NIOS simply waits. Initially, I assigned a few images to NIOS for processing parallel to the HPS. However, one image processing in NIOS takes longer than the HPS 99 image processing time which ultimately slows down the system. Therefore, NIOS is completely offloaded. The software modification on the HPS side is kept minimal other than tweaking some loops. However, since the HPS has NEON vector processing it would be faster if we make the following loop such that it can be processed by the SMID instruction.

```
float *aAdr = aPtr;
    for (int aRow = 0; aRow < aRows; aRow++) {
        float *bAdr = bPtr;
        for (int aCol = 0; aCol < aCols; aCol = aCol + 4) {
        mul1 = aAdr[0] * bAdr[0];
        mul2 = aAdr[1] * bAdr[1];
        mul3 = aAdr[2] * bAdr[2];
        mul4 = aAdr[3] * bAdr[3];
        cPtr[aRow] += mul1 + mul2 + mul3 + mul4;
        aAdr += 4;
        bAdr += 4;
        }
    }
```

Instead of using HPS, the solution could have exploited multiple NIOS and other FPGA features. This is even further justified if we look at the resource usage. This HPS based solution only consumed 10% ALM that means we have an enormous resource to make the design parallel. Also, the software could have been made faster by replacing some of the complicated functions. For example, replacing *sigmoid* with *ReLu* would liberate the processors from calculating complicated mathematical function. We have 10% margin in the detection accuracy therefore accuracy would not be a bottleneck. Finally, given enough time, the neural network itself can be improved to operate in parallel hardware.

IV. Instruction for running the design

1) Load the *.sof* file from \codesign-challenge-jubayer0175\exampleniossdram.sof using NIOS shell:
   nios2-configure-sof -d 2 exampleniossdram.sof
2) Open a NIOS terminal in another NIOS shell: *nios-terminal*
3) Load the NIOS software: cd \nios _v2_neuralnetwork\
   nios2-download main.elf –go
   NiOS 2 termianl should print this:

```
nios2-terminal: connected to hardware target using JTAG UART on cable
nios2-terminal: "DE-SoC [USB-1]", device 2, instance 0
nios2-terminal: (Use the IDE stop button or Ctrl-C to terminate)

Please initiate processing by starting HPS
```

4) Load the mnist_infertest file to the HPS from the software \hps _v2_neuralnetwork\directory.
5) Run the mnist_infertest file in the HPS. The NIOS 2 terminal should print:

```
nios2-terminal: connected to hardware target using JTAG UART on cable
nios2-terminal: "DE-SoC [USB-1]", device 2, instance 0
nios2-terminal: (Use the IDE stop button or Ctrl-C to terminate)

Please initiate processing by starting HPS
Done processing: Total cycles: 2602091
Errors: 9
```