Instructions for the MNIST-NN Reference Software

Fall 2019

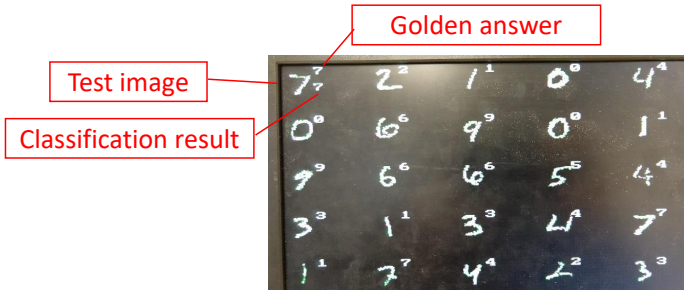
This document provides a brief overview of how to run the MNIST-NN reference SW on Altera DE2-115 boards. The SW has been derived from KANN, a lightweight neural network library written in pure C. Two different types of neural networks are given under **source/**: (i) MNIST-MLP, provided as an example network written with KANN APIs, consists of 1 hidden layer with 64 neurons and yields a 97.59% accuracy on 10000 MNIST test images. (ii) MNIST-CNN, which incorporates 2 convolution layers (32 3x3 filters, ReLU) followed by a fully-connected layer (128 neurons), yields a 99.13% accuracy on the same test image set. The scope of this project is limited to using the pre-trained model and accelerating the inference process.

The steps involved are: (i) adding the necessary peripherals to the SoC from HW2 so that the SW running on NiosII processor can access the on-board flash memory (where the pre-trained model and the test images will be stored), (ii) creating a new NiosII application project with the reference SW and configuring the BSP project to support ROZIPFS (Read-Only ZIP File System), (iii) creating an ROZIPFS image from the input files and programming the file system onto the flash memory, and (iv) building and running the application on a DE2-115 board.

Step 1: Adding peripherals to access the flash memory: The refer-

ence SW provided to you takes its input from the flash chip on the DE2-115 board (pre-trained models ***mnist-cnn.kan***, ***mnist-mlp.kan*** & the first 100 test images ***mnist-test-100.knd***), and displays the test images, their labels, and classification results on the VGA display as shown in Figure 1.

Figure 1: VGA Display



The system that you configured in HW2 already has the VGA output related hardware modules. However, you need to add additional hardware to read from and write to the 8MB flash chip on the board. To do this you need to add two components from the Qsys Component Library: a **Generic Tristate Controller** and a **Tri-State Conduit Bridge**.

Inside the configuration window for Generic Tristate Controller, you can find **Presets Library** on the right. Click on **Flash Memory Interface (CFI)** and hit **Apply**. Change the default options to those in Table 1.

After adding both modules to your system, make all the signal connections appropriately.

1. *clk* and *reset* signals are connected to *clk_0.clk* and *clk_0.clk_reset*.
2. *generic_tristate_controller_0.uas* is connected to *nios2_qsys_0.data_master*.
3. *generic_tristate_controller_0.tcm* is connected to *tristate_conduit_bridge_0.tcs*.
4. Export *tristate_conduit_bridge_0.out* by double-clicking on the **Export** tab.

Table 1:

Signal Selection Tab	
Address width	23
Data width	8
Byteenable width	1
Bytes per word	1
Signal Timing Tab	
Read wait time	160
Write wait time	160
Setup time	60
Data hold time	60
Maximum pending read transactions	3
Turnaround time	2
Timing units	Nanoseconds
Read latency	2
Module Assignments Section (in any tab)	
hwClassnameDriverSupportList	altera_avalon_lan91c111:altera_avalon_cfi_flash
hwClassnameDriverSupportDefault	altera_avalon_cfi_flash
SETUP_VALUE	60
WAIT_VALUE	160
HOLD_VALUE	60
TIMING_UNITS	ns
SIZE	8388608u
MEM_INIT_DATA_WIDTH	8
HAS_BYTE_LANE	0
IS_FLASH	1
GENERATE_DAT_SYM	1
GENERATE_FLASH	1
DAT_SYM_INSTALL_DIR	SIM_DIR
FLASH_INSTALL_DIR	APP_DIR

We'll add two more modules in order to show text (labels and classification results) on the VGA display. First, select **Character Buffer for VGA Display** from the Qsys Component Library. Set **Video-Out Device** to **On-board VGA DAC** and enable **Enable Transparency**. Make the connections below.

1. clk: clk_0.clk
2. reset: clk_0.clk_reset
3. avalon_char_control_slave: nios2_qsys_0.data_master
4. avalon_char_buffer_slave: nios2_qsys_0.data_master

Next, select **Alpha Blender** from the Qsys Component Library and set **Alpha Blending Mode** to **Simple**. Make the connections below:

1. clk: clk_0.clk
2. reset: clk_0.clk_reset
3. avalon_foreground_sink: video_character_buffer_with_dma_0.avalon_char_source
4. avalon_background_sink: video_rgb_resampler_0.avalon_rgb_source
5. avalon_blended_source: video_dual_clock_buffer_0.avalon_dc_buffer_sink

We will change the processor type from NiosII/e to NiosII/f and set it as the baseline. NiosII/f includes embedded multipliers and caches which you will try to outperform. Double-click on ***nios2_qsys_0*** and set **Nios II Core** to **Nios II/f**. In the **Caches and Memory Interfaces** tab, set **Instruction Cache** and **Data Cache** to the maximum size available (64 KB). You may adjust the cache sizes if you would like to utilize the on-chip block RAM resources for other purposes. (Note: Do not enable **Bursts transfers** as it is known to make the system hang)

If you have not added a **Performance Counter** to your system, go ahead and do it now. Finally run **System - Assign Base Addresses** and **System - Create Global Reset Network** from the top menu. If the base address of

SRAM Controller has been changed, modify the buffer start address of **Pixel Buffer DMA Controller** to match the base address of **SRAM Controller** and modify the back buffer start address accordingly.

Once you have made the above changes in Qsys, go ahead and generate the system. If you take a look at the top level Verilog file generated by Qsys (***nios_system.v***), you will see that there are additional input and output pins as compared to your previous system. These correspond to the tri-state connection peripherals that you have just added, and they need to be connected to the appropriate pins on the board. You will have to make appropriate changes to your top-level file ***demo.v***. Note that sigals **FL_RST_N** and **FL_WP_N** are hardwired to logic value '1' while **FL_RY** can be left open. Refer to the DE2-115 Board Manual and the tri-state module documentations in Qsys for further details. Once you are done, compile the project again using Quartus and program the new .sof file onto the board.

Step 2: Building and running the reference software: Now proceed to NiosII IDE and create a BSP project with 2 blank application projects for MNIST-MLP and MNIST-CNN. Add all the .c and .h files under **source/mnist-mlp/** and **source/mnist-cnn/** to their respective application projects. Read through the relevant sections of Chapter 5 of the Nios II System Architect Design manual.

1. **Specify the Zip File System Settings** describes configuring the ROZIPFS software component inside BSP Editor.

2. **Program the Zip File to Flash Memory** describes programming the .zip file to the flash. Use ***mnist-nn.zip*** provided with the source code.

A few things to note:

1. ROZIPFS only allows "uncompressed" .zip files. In order to place input files to an uncompressed .zip file on Linux, use "zip -r -0 *target_filename.zip file1.zip file2.zip fileN.zip*".

2. In **NiosII BSP Editor**, assign **ro_zipfs_base** to the base address of your Generic Tristate Controller and **ro_zipfs_offset** to 0x0. The offset of 0x0 should also be used in **Flash Programmer** as well.

3. The provided source code assumes that the ROZIP File System has been mounted at the path `/mnt/rozipfs` in the flash memory.

4. In **Nios II Flash Programmer**, click **Connections** and check **Ignore mismatched system ID** and **Ignore mismatched system timestamp** to avoid errors.

5. Search for keyword **"ECE695R"** in ***mnist-mlp.c*** and ***mnist-cnn.c*** to check the modifications made to the original SW code. Modify **NUM_TEST_IMAGES** to adjust the number of test images to be classified. You can infer the network configuration from the code written for training.

You can now build and run the reference SW. Test images will be shown on the VGA display along with their labels, and as each image is classified the result will be shown below the label.

Step 3: Profiling your SW: There are two ways to profile the application. One option is to use **Performance Counter**; make sure that your Performance Counter supports the enough number of simultaneously measured sections. Another way is to use the built-in **gprof** functionality in NiosII IDE. Refer to the Profiling Nios II Systems document for more information. A few things to note when using gprof are:

1. In Qsys, set **timeout** of **Interval Timer** to 10 seconds. Also make sure that **Avalon Memory Mapped Slave** of Interval Timer is connected to both **data_master** and **instruction_master** of the NiosII processor, and an IRQ number is assigned.

2. When using gprof, set **USE_PERFORMANCE_COUNTER** declared in ***djpeg.c*** to 0 in order to turn off the Performance Counter. Otherwise

gprof will make your program run forever. You need to get a broad sense of the function call status by using gprof only, and then measure the detailed running times of each function using Performance Counter only.

3. If your program runs correctly with gprof, you'll see ***gmon.out*** created in your application project. Right-click on the application project and select **Nios II - Nios II Command Shell**. Type the comand **"nios2-elf-gprof your-project-name.elf gmon.out >report.txt"** and open ***report.txt*** to check the profiling result.