Project: Neural network acceleration for image recognition of digits

Olivér Facklam, Ju Wu

June 2021

Contents

Introduction

The goal of this project is to create a multi-master system performing digit recognition on images, through a neural network accelerator. Image acquisition is done through a TRDB-D5M camera with a controller developed during last semester. The accelerator is implemented in VHDL and exploits burst capabilities to read the image contents, and parallelized operations for inference. A very simple neural network (constrained by FPGA resources) is implemented to classify handwritten digits 0 through 4.

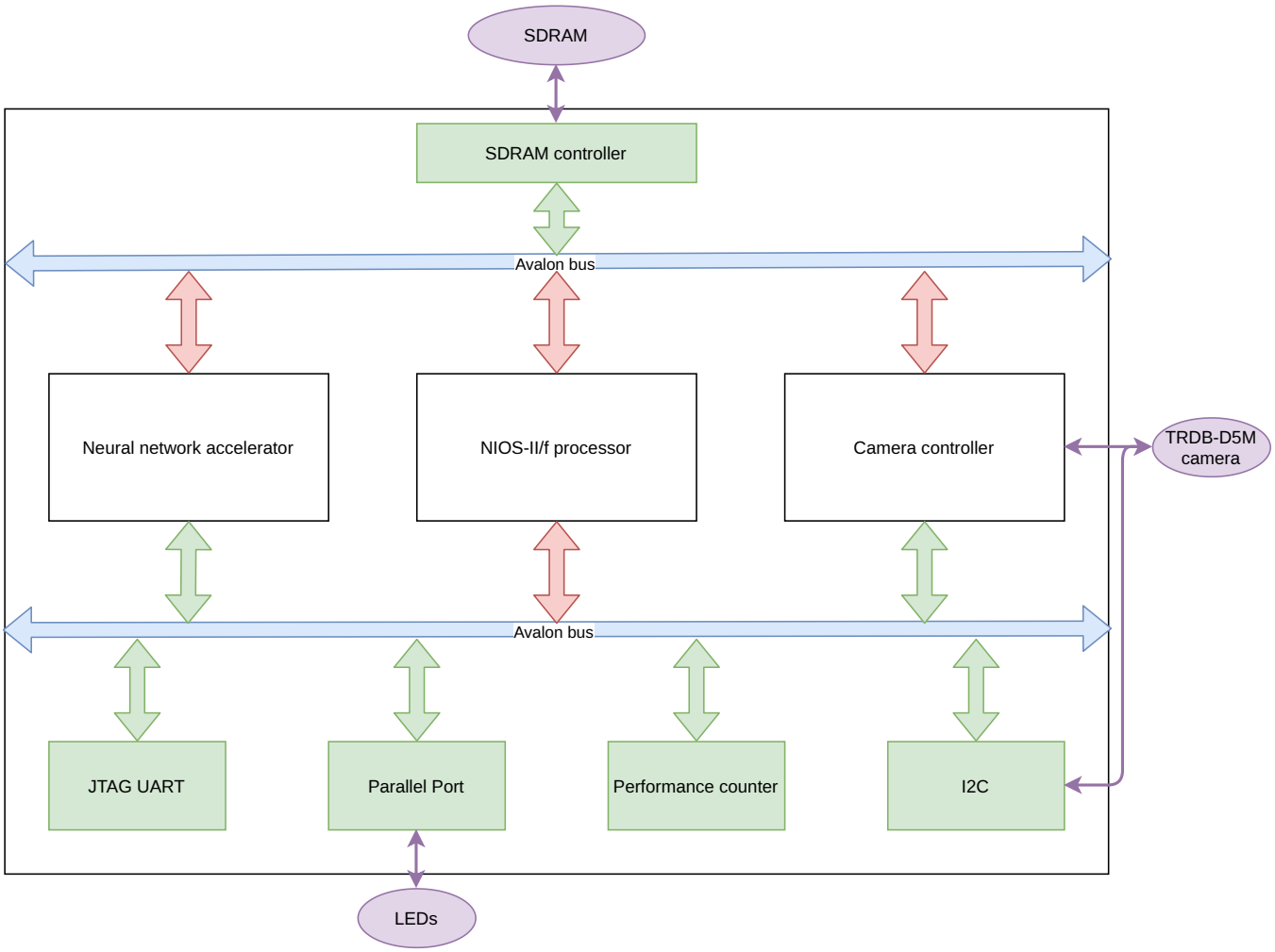

Figure 1: System architecture overview – Avalon master interfaces are in red, Avalon slave interfaces in green

The global system architecture is presented in section 1 of this report. The neural network accelerator is detailed in section 2. Section 3 & 4 respectively describe system operation and its real-time analysis. Finally section 5 presents the obtained results.

1 System Architecture

The global system architecture is shown in figure 1. The system is composed of:

- a NIOS-II/f processor executing the main program code and responsible for image transformation,
- a 64 MB SDRAM storing the program code, as well as the raw captured image, the transformed image and some of the neural network's weights,
- a camera controller for TRDB-D5M, developed during last semester, allowing to capture a frame from the camera on demand, and storing it directly into the SDRAM through the integrated DMA unit,
- an I^2C interface, taken from Moodle, to set up the camera,
- a neural network controller – detailed in the next section,
- a parallel port connected to the LEDs to display the inference result,
- a JTAG interface and a performance counter, for debugging & profiling.

2 Neural Network Accelerator

The neural network accelerator is composed of 3 major blocks, as displayed on figure 2:

- the neural network itself, taking as inputs the layer weights and input data, and outputting an inferred vector,
- an argmax module, converting the output vector of the neural network into a one-hot encoded vector (i.e. there is a 1 at the index with the highest output value),
- a DMA unit supporting burst reads, to efficiently retrieve the image input, as well as part of the network weights.

There are a couple of things to note here. First, all computation inside the neural network accelerator is done using fixed-point arithmetic. Values are encoded using a 16-bit representation, with the top 8 bits being the integer part, and the bottom 8 bits the fractional part of a number. This representation works well for our purposes – the only subtle point that needs to be taken into account is shifting the result of a multiplication correctly. Saturation is not handled properly, values will overflow / underflow by wrapping around. In practice however the weights are very small, so an overflow is unlikely.
Second, the resource constraints on the FPGA caused major issues during the design phase. Indeed the problem had to be scaled down to only recognize digits 0 through 4, instead of the initially planned 0 through 9. The input image resolution was also scaled down from 28x28 to 16x16 pixels, and the number of layers was scaled down from 3 to 2. Even like this, all weights of our 2 fully connected layers could not fit into the FPGA. Thus, the weights of the input layer have to be streamed in through the DMA alongside the input data.

The accelerator's main finite state machine waits in an idle state until it receives a start signal through the Avalon slave interface. It then sends a start command to the DMA unit, and simultaneously also resets the values accumulated in the neural network. Finally it waits for the DMA unit to finish its reading process, before returning into the idle state. The result of the computation is made available on the slave interface, alongside a "busy" bit indicating the module's status.

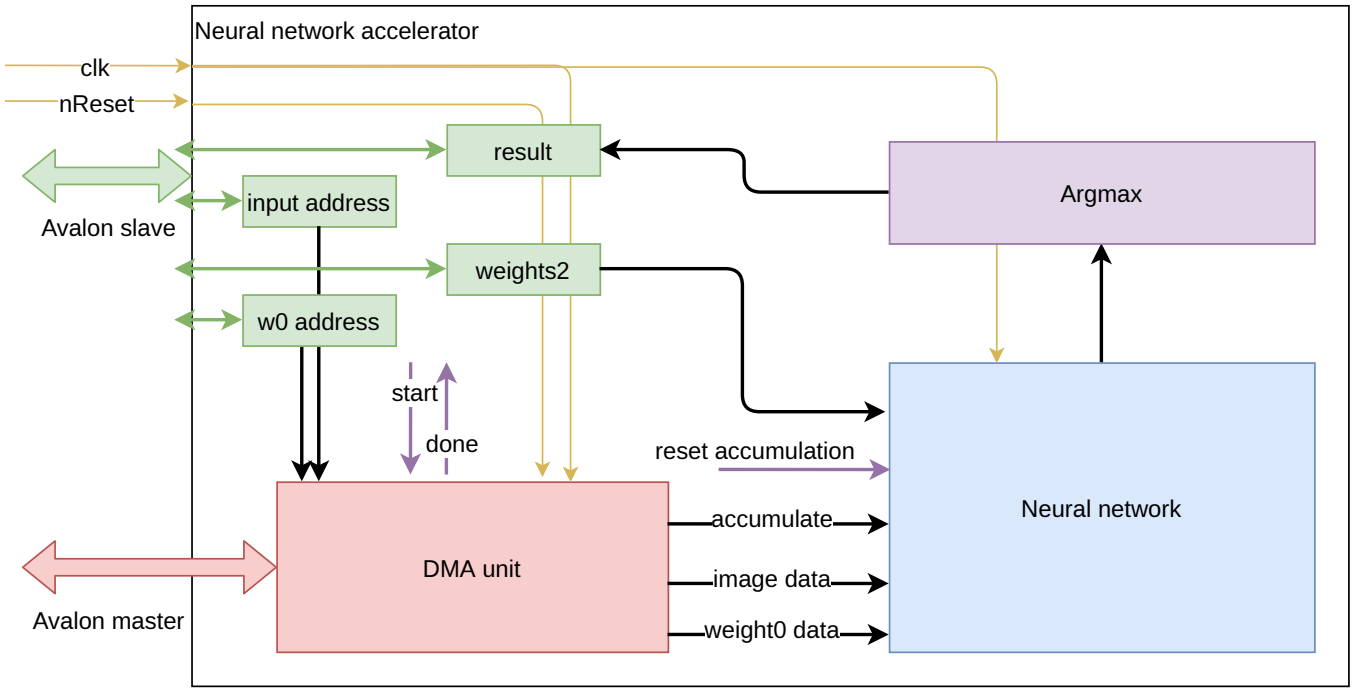

Figure 2: Architecture of the neural network accelerator

2.1 Register map

Table 1 presents the register map of the accelerator.

Address	Register
0x0	**Write:** start signal – **Read:** busy bit & result vector
0x1	Address for layer 0's weights
0x8	Row (top 16 bits) & column (bottom 16 bits) indexes for a future write to 0x9
0x9	Weight value for layer 2 that goes into the cell indexed by the values in 0x8
0xc	Address for the input data

Table 1: Register map of neural network Avalon slave

Addresses 2 to 7 and 10 - 11 were initially used for bias vectors as well as an additional "layer 1" but are currently unused since the design became smaller.

2.2 Neural network

The neural network is composed of one input layer ("layer 0") and one regular layer ("layer 2"). The characteristics of the implemented network are shown in figure 3. The various sub-entities (layer, neuron) are implemented in a generic manner, but the overall network entity is specialized to this specific neural net.

The input layer receives the image data pixel by pixel, the weights column by column, and an accumulate signal telling it to add the current inputs to its state. In practice, an accumulate pulse will be generated every time a pixel and its corresponding weight column have been loaded. Once the values for all pixels have been accumulated, the output vector contains the complete information corresponding to the input image. This is what is fed to the next layer. The accumulated values are reset every time we start processing a new image frame.

The regular layer is a simple combinational circuit, computing the matrix multiplication of the weights with the input vector. This produces a 5-dimensional output vector which is sent to the argmax module.

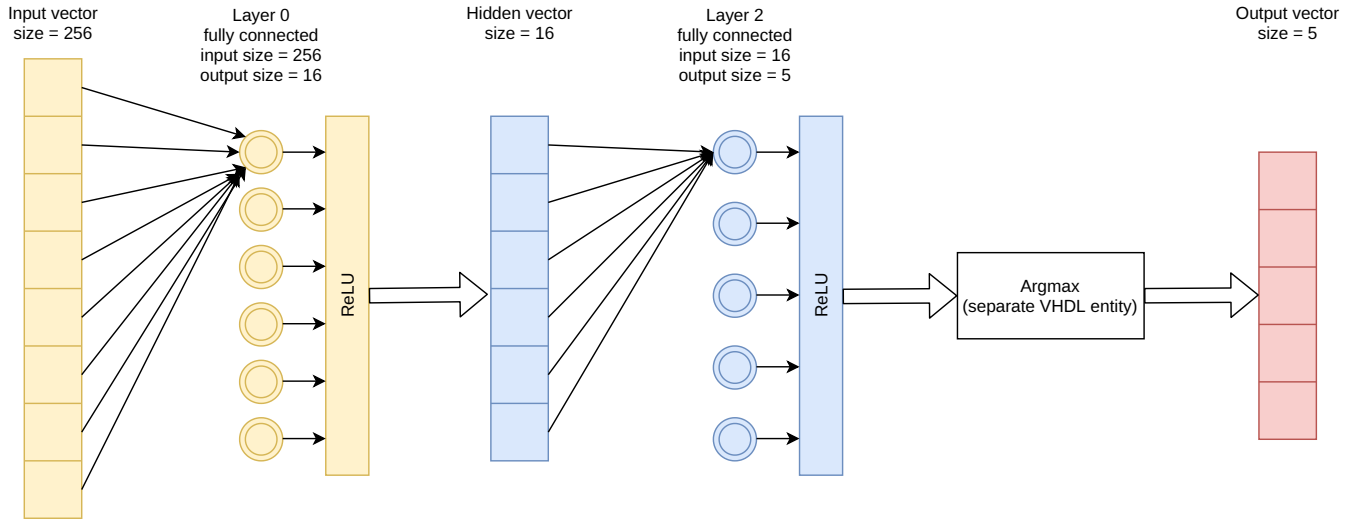

Figure 3: Implemented neural network

2.3 DMA unit

The DMA unit operates on a pixel-by-pixel basis. At each iteration, it will load one pixel of image data, followed by the corresponding column of weights loaded through a burst of 16 words. Once such an iteration is done and all the data relevant to that pixel are available, an accumulate pulse is generated before moving to the next iteration.

3 System operation

The system operates in an infinite loop. Each loop iteration corresponds to the processing and analysis of a single frame. Synchronization between the three masters is achieved through the sequential nature of this processing pipeline: all operations are managed (started and awaited) by the code executed on the NIOS-II processor.
Each loop iteration contains the following elements:

1. The processors sends a start command to the camera controller (and waits for the acquisition to be finished).
2. The camera controller acquires a camera frame and stores it into a dedicated buffer in the SDRAM. It notifies the processor through a status register.
3. The processor pre-processes the acquired frame by doing some image transformations: rotating, resizing, saturating,... The resulting values are stored in a seperate buffer in the SDRAM.
4. The processor sends a start command to the neural network (and waits for the inference to be finished).
5. The neural network accelerator loads the input image (along with the input layer's weights) and accumulates/computes the result. It notifies the processor through its status register.
6. The processor retrieves the inferred vector from the neural network's registers. It sets the parallel port to output this vector to the LEDs.

4 Real-time analysis

We would like our system to react in less than 500 ms. Let us try to estimate what the processing time of a new image could be.

Image acquisition time First, in the worst case two frames need to be awaited before having a frame ready. Indeed, one frame might just have started when the camera controller receives the starting command. In this case, the controller needs to wait for this frame to be done, and then needs to sample the next frame. This incurs a maximum delay of $t_{\text{frame sample}} = 50ms$ when considering a frame rate of 40 fps.
It is much more difficult to estimate the maximum time it takes to write the frame to memory. The SDRAM documentation claims that while doing bursts, the access time can be as low as a single cycle. However, setting up such a burst transfer costs a quite long time, which we will estimate here to 100 μs. Since the camera controller does 480 bursts of 80 words, the total transfer time is about $t_{\text{frame transfer}} = 49ms$.
In total, this makes for a worst case image acquisition time of roughly 100 ms.

Image transformation time Let us suppose that just like before, reading the raw image and writing the transformed image take respectively around 50 ms. The transformation itself can take up to 100 cycles per pixel, meaning a total time of 150 ms. This means that the transformation stage takes roughly 250 ms.

Neural network inference To read the input and weights, the accelerator does one regular access and one 16 word burst access for each of the 256 pixels. This gives a load time of around 52 ms, considering previous assumptions. Once the data is loaded, the inference itself is done in a single cycle, the rest of the circuit being combinational.

Overall, this analysis shows that the processing of a new frame can take around 400 ms, which would respect our real-time constraints of 500 ms. Of course, this quick analysis depends on many assumptions which would need to be checked. The next section presents quantitative results from experiments on the real system.

5 Results

The obtained system works! Although sometimes the output is a little unstable, and quite sensitive to the placement of the input digits, overall the system can usually generate the correct LED pattern for a handwritten digit placed in front of the camera. The behavior can be observed in the demo video available next to this report.

Some profiling was done to evaluate quantitatively the performance of our system. For this, a hardware performance counter was used. The image acquisition, image transformation, and neural network inference sections were tracked for a total of 100 iterations. Table 2 presents the corresponding results.

Section	Time/iteration (ms)	Time/iteration (cycles)
Image capture	96.5	$4.83 \cdot 10^6$
Image transformation	181	$9.03 \cdot 10^6$
Neural network inference	0.47	$2.34 \cdot 10^4$
Total	278	$13.9 \cdot 10^6$

Table 2: Performance of the 3 main sections of system operation (with hardware accelerator)

As we can observe, the neural network inference represents a tiny fraction of the total processing time, while image capture and image transformation are dominating. Image acquisition could be improved, but is constrained by the frame rate of the camera. In contrast, the image transformation could be significantly sped up by having some dedicated hardware (separately or integrated into the camera controller) instead of doing this by software. In total, the processing of a single frame takes around 280 ms, which respects our real-time constraints. It even performs somewhat better than our rough estimate from the previous section.

For comparison, the profiling was also done with a full software solution, which evaluates the neural network directly with the help of the processor's floating-point unit. Table 3 shows the profiling results.

Section	Time/iteration (ms)	Time/iteration (cycles)
Image capture	59.6	$2.98 \cdot 10^6$
Image transformation	181	$9.04 \cdot 10^6$
Neural network inference	36.7	$1.84 \cdot 10^6$
Total	277	$13.9 \cdot 10^6$

Table 3: Performance of the 3 main sections of system operation (full software)

The first thing to observe is that the neural network inference in software takes around 75× more time than the hardware accelerator. This shows that the inference is significantly sped up by the custom hardware we have developed.
However, we can also see that the total iteration time is roughly the same. The image capture takes longer when using the hardware solution, probably because of some form of synchronization happening when waiting for a new camera frame to became valid.

Conclusion

In this project, we have developed a working multi-master system, capable of classifying handwritten digits using a hardware neural network accelerator. The result is somewhat unstable, but overall the system can correctly recognize certain digits and display the corresponding output on the LEDs, as shown in the attached demo video.
Even though the hardware accelerator doesn't improve the total processing time for a given image frame with respect to a full software solution, the neural network inference itself is sped up by a factor of 75×. The process could further be improved by having dedicated hardware to perform the image transformation.