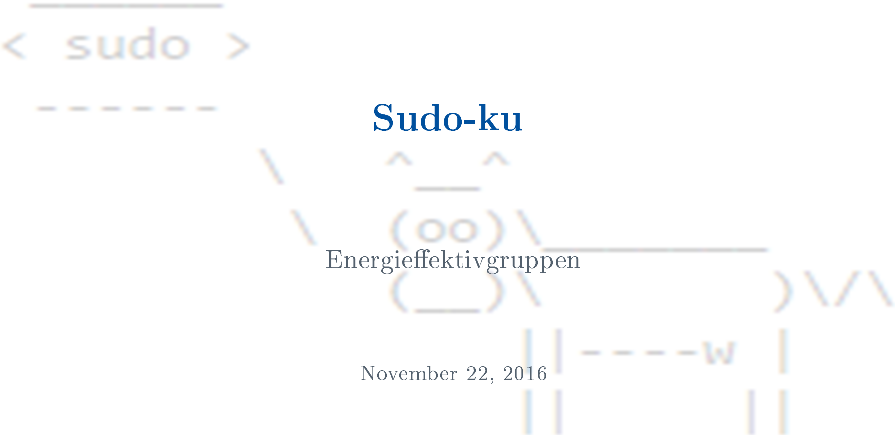


Sudo-ku

Energieffektivgruppen

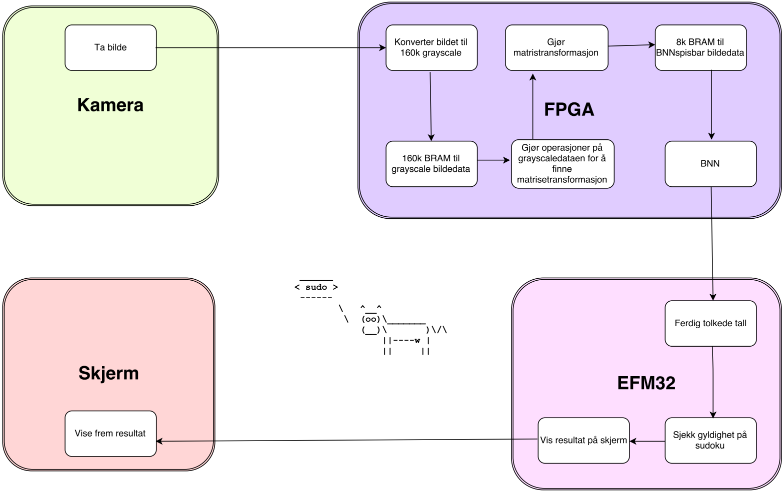
November 22, 2016

Content

Planned operation of sudo-ku

- Point camera at an image of a solved sudoku-board
- Push button
- Camera image is stored in FPGA BRAM
- FPGA figures out the transform required to rotate, scale and crop the input image
- FPGA performs transform, cutting it into pieces, trimming edges of the squares to avoid sending outlines into the BNN etc.
- BNN (on FPGA) detects digits
- Digits are sent to the MCU, where the sudoku is checked and the result is output to the user

Planned operation of sudo-ku

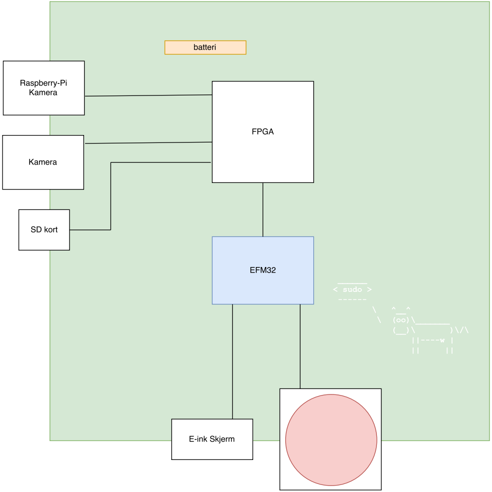


Planned operation of sudo-ku

Picture requirements

- All but the board itself should be black, the board itself white
- The camera must be pointed roughly orthogonally to the board; we do not do any perspective transformations
- The camera can be rotated along the camera-pointing axis by 45 degree either way. This requirement might go away if we find a simple way to figure out the orientation without it.

Design Overview



Argumentation

Camera (and SD-card) connected to FPGA

- We need 9x9 times 28x28 pixels of relevant image data. Results in 9x9x28x28x2 = 127008 pixels.
- Smallest standard accommodating this is HVGA(480x320=153600 pixels). But is uncommonly supported. Hence we use VGA(640x480=307200 pixels).
- We use 1 bit black/white to store the images. Thus we need 307200 bitsof storage.
- Image from the the camera is in a different format than our 1-bit black/white. This needs conversion on the fly.
- Considering the size of the image, it is possible to do keep the image in the memory of the FPGA without the use of an external SRAM

Argumentation
Image transformation on FPGA

- With the restrictions we put on the picture we can find the corners with a somewhat naive algorithm that should be possible to implement on the FPGA
- The process of doing an affine transformation on the camera input data is a simple, arithmetical operation, easy to implement elegantly on an FPGA.
- The resulting, transformed image/images are only needed on the FPGA

Argumentation
User I/O on MCU

- Output, which includes communicating with a display, is a lot more difficult implementing in hardware, with no real benefit.
- The MCU is used to control the operation flow of *sudo-ku*.

Argumentation

Sudoku-checking on MCU

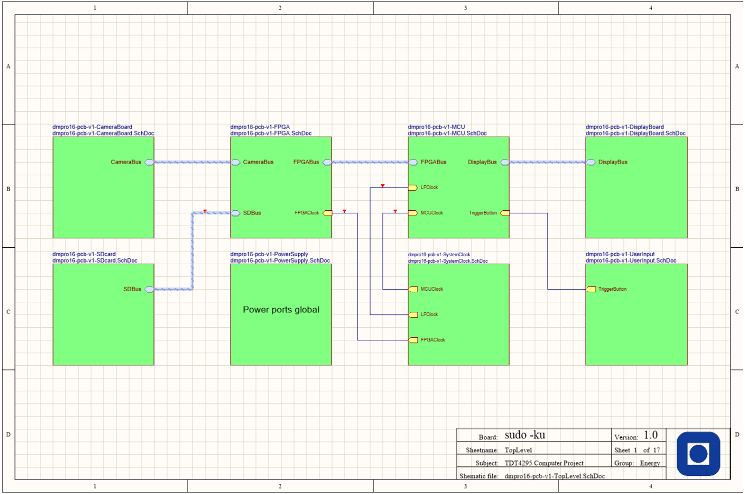
- Super easy to do on MCU.
- Have already made a first draft of about 50 lines of code checking a sudoku.
- Lots of eye-candy can be implemented when everything works
- Only 700 ways to solve a sudoku. The entire board can be transferred from the FPGA to the MCU in 10 bits. Data transferred is negligible.

Argumentation

Why we need SD-card

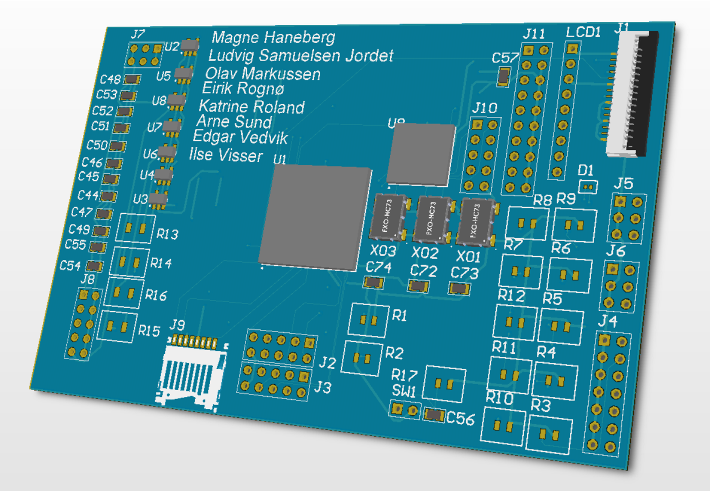
- In case the camera processing does not work, we will still have a way to demo the working BNN by loading pre-formatted, square image data, skipping the transformation steps.

PCB Design
Top level schematic



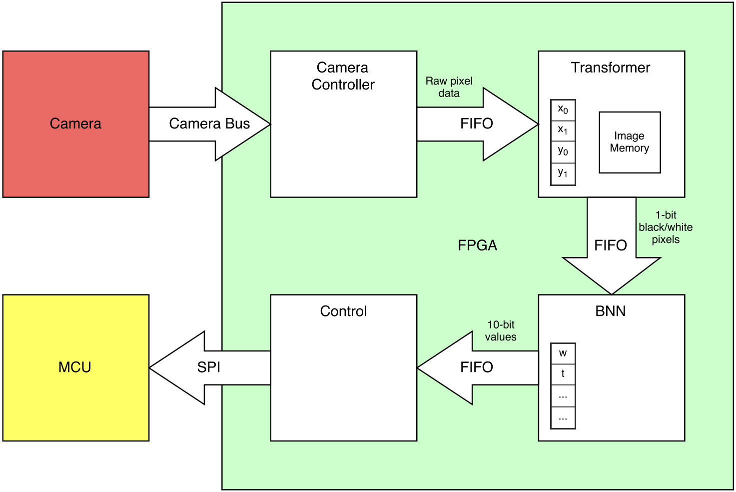
PCB Design

3D Drawing



Data flow and memory

On the FPGA



Data flow and memory

In the Transformer-module

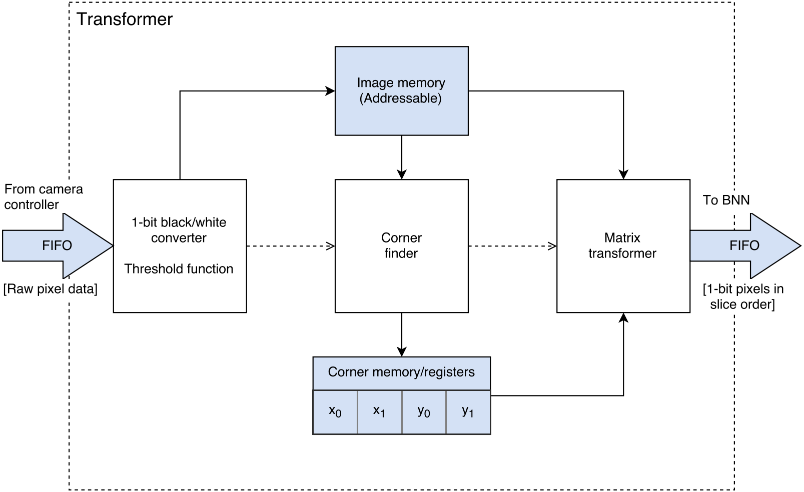


Image transformation
Cropping and straightening

- Done by a matrix transform *from* the required destination image coordinates (pixels) *to* input-image coordinates (sub-pixel)
- Uses coordinates of the topmost corners

Image transformation

Cropping and straightening

- Done by a matrix transform *from* the required destination image coordinates (pixels) *to* input-image coordinates (sub-pixel)
- Uses coordinates of the topmost corners

$$T(x, y) = \begin{bmatrix} a & -b \\ b & a \end{bmatrix} \cdot \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} x_0 \\ y_0 \end{bmatrix}$$

$$a = \frac{x_1 - x_0}{w_{\text{dest}}} \qquad b = \frac{y_1 - y_0}{w_{\text{dest}}}$$

Image transformation

Slicing and padding

- A *slice* is a small square containing one digit
- The pixels are to be sent into the BNN slicewise

Image transformation
Slicing and padding

- A *slice* is a small square containing one digit
- The pixels are to be sent into the BNN slicewise
- Going from slice coordinates (n, x', y'), n being the slice number, to sudoku-wide coordinates (x, y) with padding w_p:

$$x = (28 + 2w_p)(n \bmod 9) + x' + w_p$$
$$y = (28 + 2w_p)\lfloor n/9 \rfloor + y' + w_p$$

- A *slice* is a small square containing one digit
- The pixels are to be sent into the BNN slicewise
- Going from slice coordinates (n, x', y'), n being the slice number, to sudoku-wide coordinates (x, y) with padding w_p:

$$x = (28 + 2w_p)(n \bmod 9) + x' + w_p$$
$$y = (28 + 2w_p)\lfloor n/9 \rfloor + y' + w_p$$

- This makes $w_{\text{dest}} = 28 \cdot 9 + 2 \cdot 9 \cdot w_p = 252 + 18w_p$

Binarizing the neural network
Changing to -1 and 1

- We can represent most of the neural network as -1 or 1 without much accuracy loss
- The input 0-255 can be -1 if $\leq$ 127 else 1
- New weights become sign(old weights)
- The activation function tanh(x) can also be replaced by sign(x)
- Now only the batch normalization is not represented by -1 and 1 yet

Binarizing the neural network

From multiplication to XNOR

	-1	1
1	-1	1
-1	1	-1

Multiplication

	0	1
1	0	1
0	1	0

XNOR

Binarizing the neural network

Finding the batch normalization threshold

- $y = \gamma\sigma^{-1}(x - \bar{x}) + \beta$
- $y = \gamma\sigma^{-1}(2x - \bar{x} - prev) + \beta$
- Find where the output is 0, so we can use only one value
- $x = 0.5(\bar{x} - \frac{\beta}{\gamma\sigma^{-1}} + prev)$

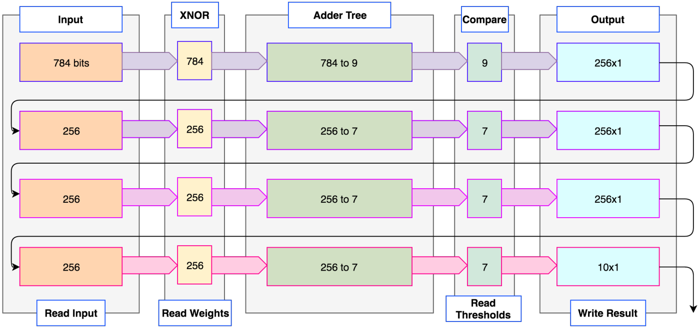
Binarizing the neural network

The binarized result

- $\sum w \otimes out$
- 0 if the sum is below the batchnorm threshold
- 1 otherwise

BNN Architecture

Image of the architecture



BNN Architecture
Pipelining

- The next neuron in one layer can start processing one clock cycle after the current neuron
- We can start on the next image as soon as the previous one is done with the first layer
- All layers can be used at the same time, each working on a different image