

< sudo >

Sudo-ku

Energieeffektivgruppen

October 9, 2016



Planned Operation

Design Overview

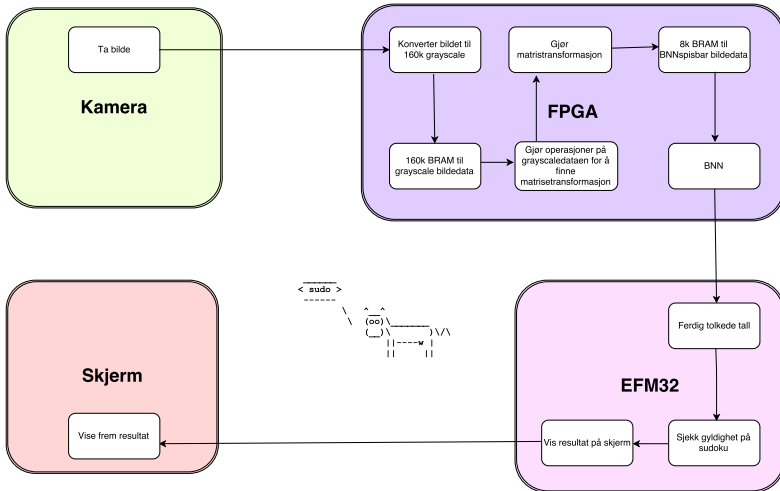
Argumentation

Binarizing the neural network



- ▶ Point camera at an image of a solved sudoku-board
- ▶ Push button
- ▶ Camera image is stored in FPGA BRAM
- ▶ MCU-program figures out the transform required to make a 289 by 289 image from 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 sent to MCU, where the sudoku is checked and result (and potentially other relevant info, incorrect rows etc.) is output to the user

Planned operation of sudo-ku



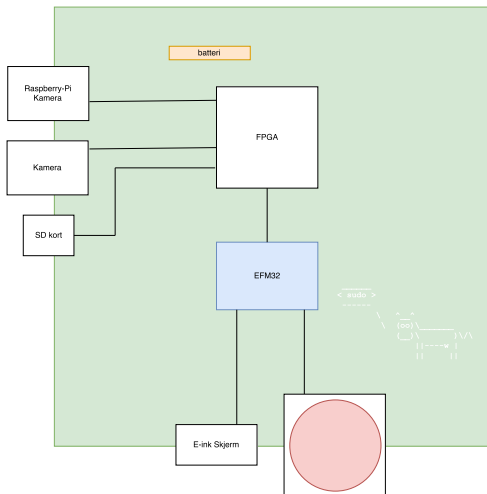
Planned operation of sudo-ku

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





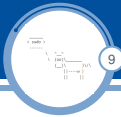
- ▶ We need 9x9 times 28x28 pixels of relevant image data. Results in $9 \times 9 \times 28 \times 28 \times 2 = 127008$ pixels.
- ▶ Smallest standard accommodating this is HVGA(480x320=153600 pixels). But is uncommonly supported. Hence we use VGA(640x480=307200 pixels).
- ▶ We use 4 bit grayscale to store the images. Equals $307200 \times 4 = 1228800$ bits.
- ▶ Image from the the camera is in a different format than our 4-bit grayscale. This needs conversion on the fly, either on FPGA or MCU.



- ▶ The process of doing an affine transformation on the camera input data is a simple, arithmetical operation, easy to implement elegantly on an FPGA, given six matrix elements.
- ▶ The resulting, transformed image/images are only needed on the FPGA



- An algorithm for figuring out where the board is located in the image is typically a task which is much more elegant (and less error-prone) implemented in software than in hardware. It will also be possible to compute with only a subset of the input pixels, thus not needing all of the pixels to be transferred even though we do the computation on the MCU and the image is stored on the FPGA.



- ▶ This is a no-brainer
- ▶ 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*.



- ▶ 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.



- ▶ 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.

Binarizing the neural network

Changing to -1 and 1



- ▶ We can represent most of the NN as -1 or 1 without much accuracy loss
- ▶ The input 0-255 can be -1 if ≤ 127 else 1
- ▶ New weights become $\text{sign}(\text{old weights})$
- ▶ The activation function $\tanh(x)$ can also be replaced by $\text{sign}(x)$
- ▶ 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