System-on-Chip (SoC) Design

ECE382M.20, Fall 2023

Homework #1

Assigned: August 31, 2023
Due: September 14, 2023

Instructions:

- Please submit your solutions via Canvas. Submissions should include a single PDF with the writeup and single Zip or Tar archive for source code.
- You may discuss the problems with your classmates but make sure to submit your own independent and individual solutions.

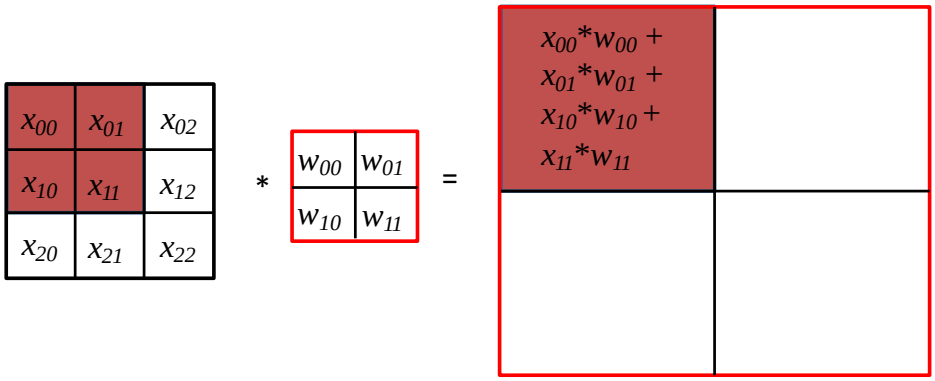
Problem 1: Convolutional Neural Networks (50 points)

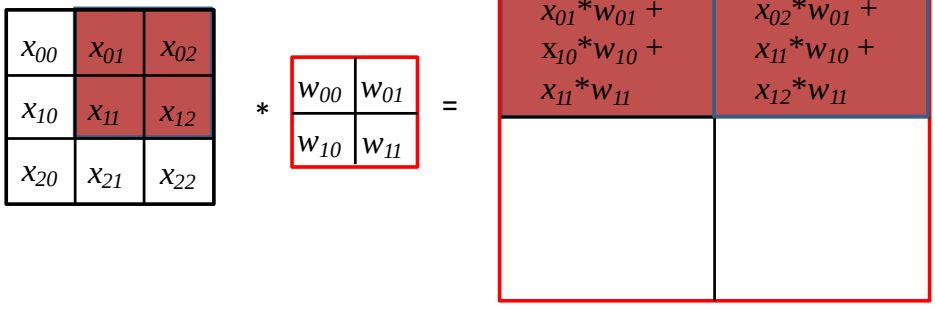
CNNs uses convolution operations primarily to extract features from the input image. We use this exercise to get familiar with how convolutions work. A convolution is done by multiplying a pixel's and its neighboring pixels color value by a filter/kernel matrix. Consider a 3x3 image and a 2x2 kernel weight matrix, whose pixels and elements are shown below:

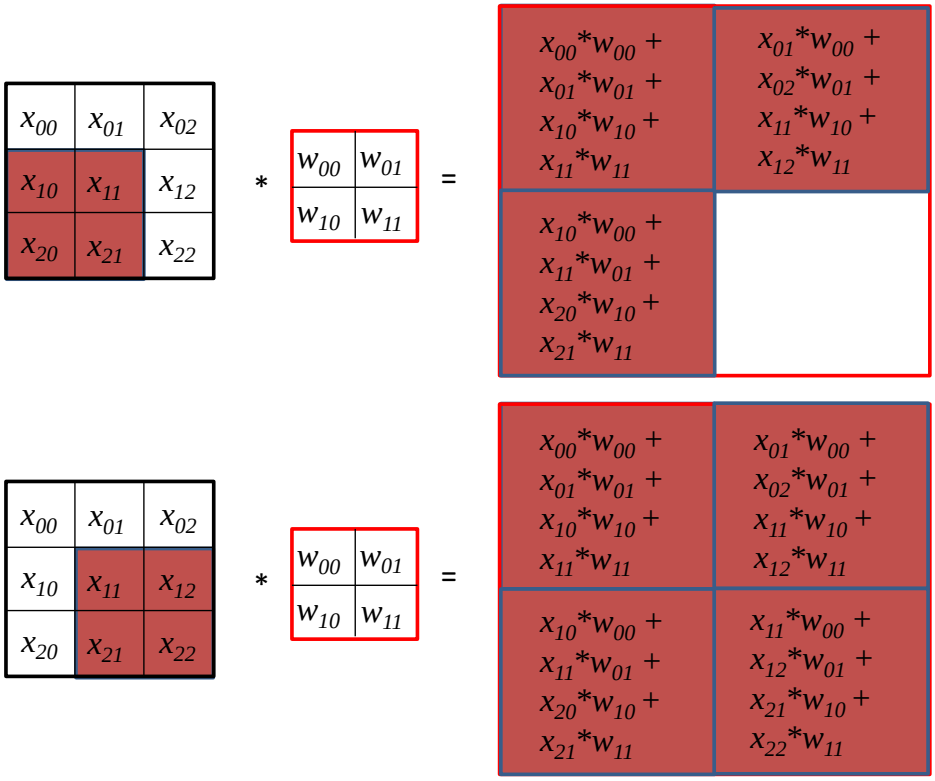
x_{00}	x_{01}	x_{02}
x_{10}	x_{11}	x_{12}
x_{20}	x_{21}	x_{22}

w_{00}	w_{01}
w_{10}	w_{11}

Then, the convolution of the 3x3 image and the 2x2 kernel can be computed as shown below:

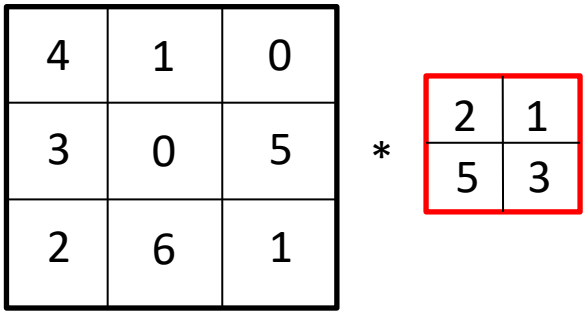






Take a moment to understand how the computation above is being done. We slide the 2x2 kernel matrix over our 3x3 image by a 1 pixel stride, and for every position, we compute the elementwise dot product to get a single element of the output matrix. Note that the 2×2 filter matrix "sees" only a part of the input image in each stride.

a) Now given the following concrete image and kernel matrix, calculate the convolution result:

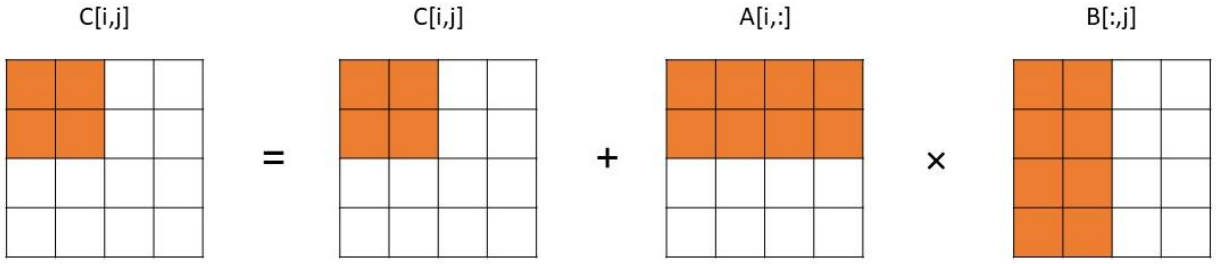


b) As discussed in class, such a convolution operation can be done by transforming it into a general matrix-matrix multiplication (GEMM). Show how this transformation and re-arrangement is performed on the example in a). What would be the matrix A and matrix B to be multiplied? Explain and draw figures as necessary. What are the pros and cons of casting the convolution as GEMM?

c) Now assume an input image I with 1 channel of general dimensions I_wxI_w, and two filters F_1 and F_2 each with 1 channel and dimensions F_wxF_w. Assuming no zero-padding and a stride of

1, what are the dimensions of the output feature map *O*? Write down the pseudocode to first transform *I*, F_1 and F_2 into A and B, and then perform the GEMM of A and B such that the output feature is calculated. What are the dimensions of A, B and C, where C is the result of the GEMM?

d) GEMM is one of the most fundamental operation in most applications in machine learning and beyond. As discussed in class, GEMM is inherently compute-bound with ample parallelism and locality. However, with the large matrix sizes, performance is often limited by on-chip memory size and/or memory bandwidth. In order to maximize data reuse, so-called blocked matrix-multiplication algorithms are often used. Shown below is the case of blocking a 4x4 matrix multiplication using a block size of 2:



Write down a pseudocode for performing a simple blocked matrix multiplication, assuming a block size *b* and square matrices A and B, both with dimensions equal to *NxN* (where *N*>>*b* and *N*%*b*=0). Feel free to use any blocking technique you like. In general, many blocking algorithms have been studied depending on the dimensions along which matrices are blocked, and their relative performance is correlated to the underlying memory architecture, e.g., cache hierarchy and configurations in a CPU.

Problem 2: SystemC (50 points)

An installation of the SystemC class library is available for you to work with on the department's LRC Linux servers (see https://wikis.utexas.edu/display/eceit/ECE+Linux+Application+Servers). Alternatively, you can install SystemC yourself locally on your own machine (see http://www.accellera.org/downloads/standards/systemc). To work with SystemC on the LRC machines, first setup the SystemC environment by loading the corresponding module as follows:

```
% module load systemc/2.3.4
```

You can then access the SystemC installation via the '$SYSTEMC' environment variable.

a) The SystemC installation comes with a set of examples, available under $SYSTEMC/examples. Create a working directory and create a local copy of the included simple FIFO example:

```
% mkdir work
% cd work
% cp -r $SYSTEMC/examples/sysc/simple_fifo .
% cd simple_fifo
```

Edit the Makefile to point the include statements to the correct path and to provide the required compile and build settings. The Makefile should look like this:

```
include $(SYSTEMC)/examples/build-unix/Makefile.config

SYSTEMC_HOME = $(SYSTEMC)
TARGET_ARCH  = linux64
FLAGS_STRICT += -Wno-variadic-macros

PROJECT := simple_fifo
SRCS    := $(wildcard *.cpp)
OBJS    := $(SRCS:.cpp=.o)

include $(SYSTEMC)/examples/build-unix/Makefile.rules
```

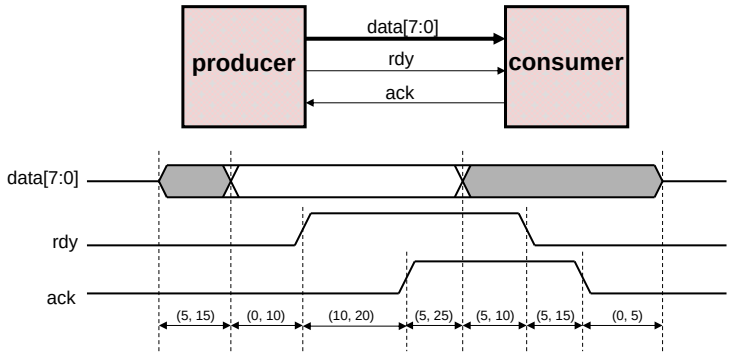
Finally, compile and run the example:

```
% make
% ./simple_fifo.x
```

Study the code and observe the program output. You can also use your favorite debugger (e.g., using `ddd` as a graphical frontend for `gdb`) to walk through the example. Briefly explain the program behavior and output, e.g. by drawing a trace of SystemC thread executions.

b) Modify the example to replace the custom FIFO channel with a corresponding `sc_fifo<char>` channel from the SystemC standard channel library. Compile and simulate the code to verify correctness. Vary the FIFO depth/size and observe the resulting behavior. Briefly explain any differences in behavior you see as a function of FIFO size/depth.

c) Now replace the `sc_fifo<char>` FIFO queue channel with `sc_signal<char>` and `sc_signal<bool>` channels that model the wires of a simple timed double handshake protocol connecting the *producer* and *consumer* modules, e.g. following this timing diagram (where tuples indicate minimum/maximum allowed times between events on the wires):



Insert code into the *consumer* that prints the current simulation time every time a character is received. Compile and simulate your design to verify its correctness and report on the simulation output.

d) Now go back to the original custom FIFO design from a). Can you modify this design by inserting time `wait()` statements into the custom queue channel to match the timing, behavior and output of the design from c)?