System-on-Chip (SoC) Design

EE382M.20, Fall 2018

Homework #1

Assigned: September 4, 2018
Due: September 20, 2018

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.

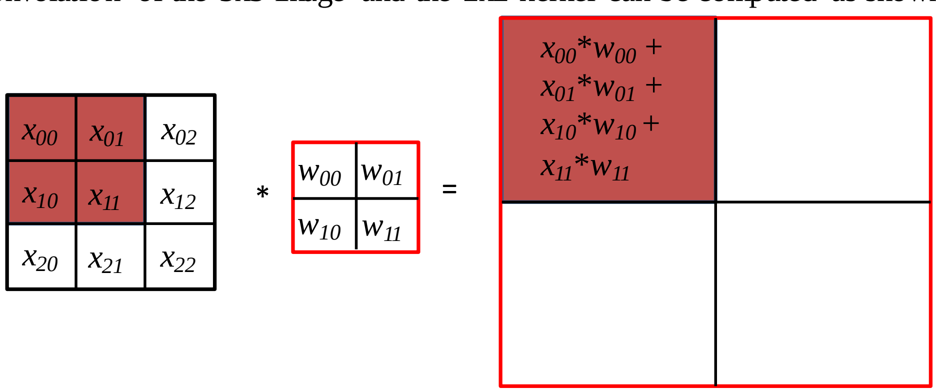
Problem 1: Convolutional Neural Networks (50 points)

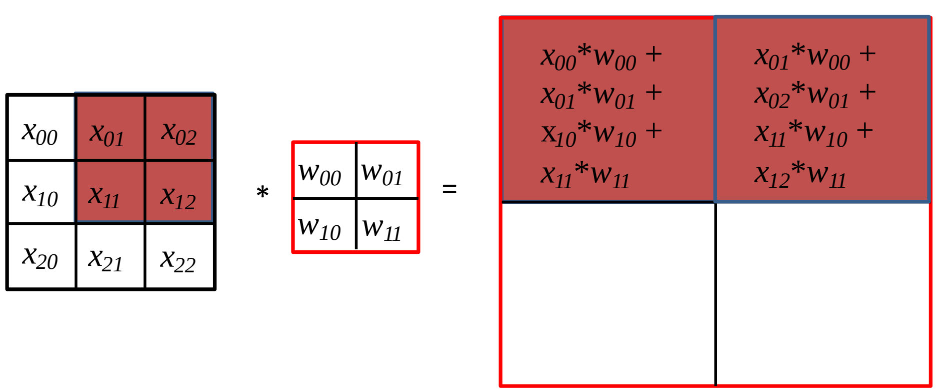
CNNs uses convolution operations primarily to extract features from the input image. We use this exercise to get familiar with how convolutions work. A convolution is done by multiplying a pixel's and its neighboring pixels color value by a filter/kernel matrix. Consider a 3x3 image and a 2x2 kernel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

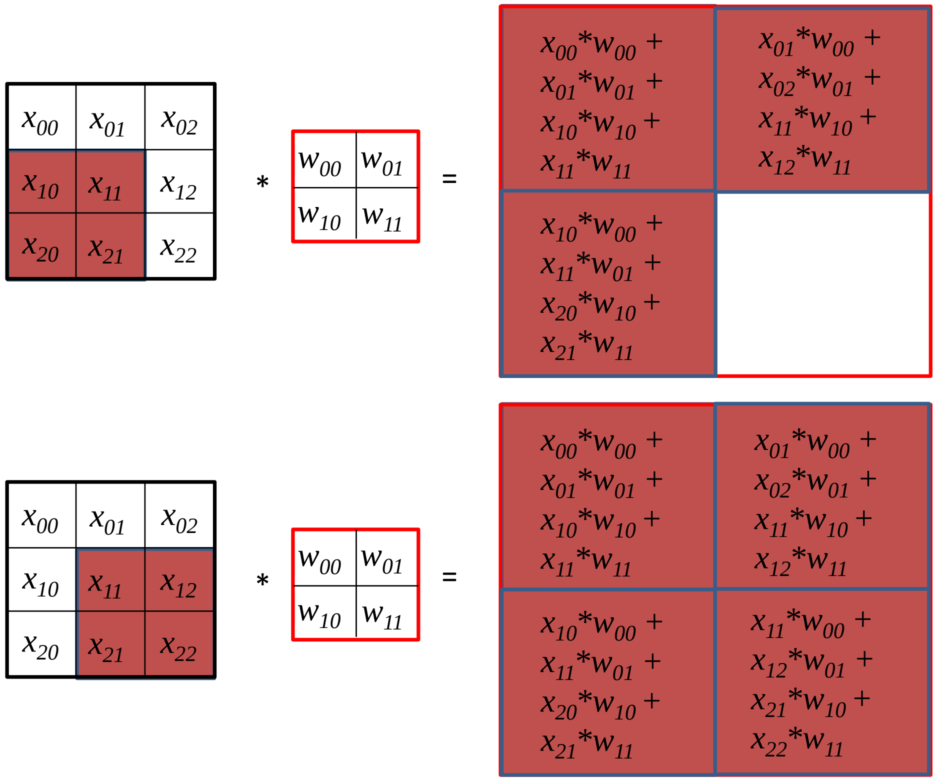
x_{00}	x_{01}	x_{02}
x_{10}	x_{11}	x_{12}
x_{20}	x_{21}	x_{22}

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:

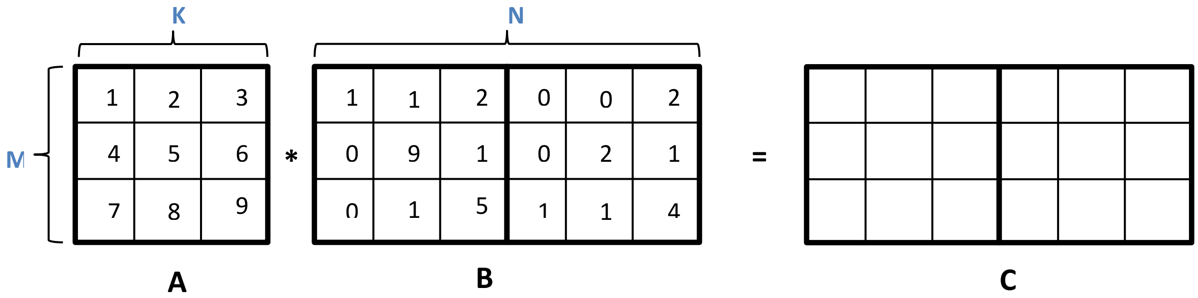
1	0	5
2	1	3
0	5	4

$*$

1	3
4	1

b) As discussed in class, such a convolution operation is usually 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.

c) Now assume that we have a cache with 120 bytes capacity, where each cache line is 8 bytes and each element of a matrix corresponds to a unique cache line. The cache is initially empty and uses an LRU for replacement policy with write-back. Given the following matrices:



Calculate the cache hit rate of the following two different matrix multiply algorithms. You can assume that variables *i*, *j* and *k* are stored in registers:

```
gemmA: for (int i=0; i< M ; i++)
           for(int j=0; j< N; j++)
                 for(int k=0; k < K; k++)
                      C[ i ][ j ] += A[ i ][ k ] * B[ k ][ j ]

gemmB: for (int j=0; j< N ; j++)
           for(int i=0; i< M; i++)
                 for(int k=0; k < K; k++)
                      C[ i ][ j ] += A[ i ][ k ] * B[ k ][ j ]
```

Explain the behavior and your observations. Can you improve the above code to increase the cache hit rate further?

Problem 2: SystemC (50 points)

To work with and develop code in SystemC, log into one of the ECE Department's LRC machines (see http://www.ece.utexas.edu/it/remote-linux) and setup the SystemC environment as follows:

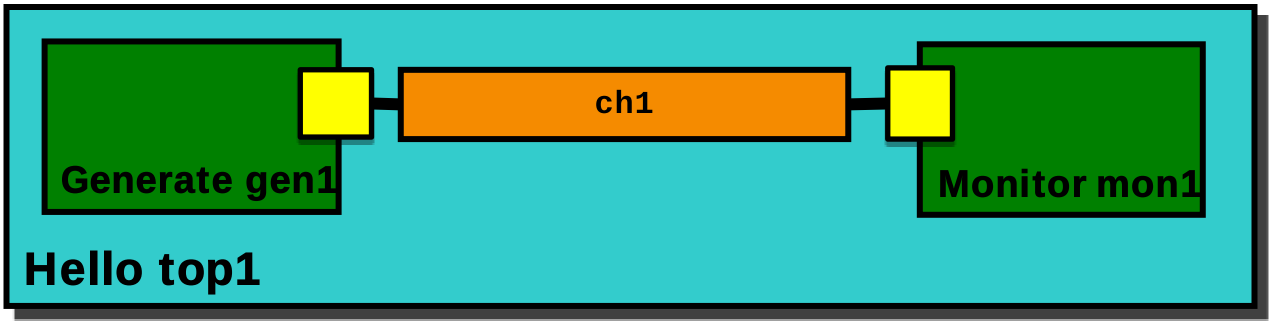
- [t]csh:
  ```
  setenv SYSTEMC /usr/local/packages/systemc-2.3.1
  setenv LD_LIBRARY_PATH $SYSTEMC/lib-linux
  ```
- [ba]sh:
  ```
  export SYSTEMC=/usr/local/packages/systemc-2.3.1
  export LD_LIBRARY_PATH=$SYSTEMC/lib-linux
  ```

You can then access the SystemC installation by referring to the '$SYSTEMC' variable.

d) Get the attached *Hello* example running: Unpack the archive, change into the `Hello-1` subdirectory, compile the example by running '`make`' and using your favorite debugger (e.g., using `ddd` as a graphical frontend for `gdb`), walk through the behavior of the example.

b) Create a for-loop in the process to output the "Hello" message 10 times in bursts with a random delay between messages evenly distributed from 50 to 90 ns.

c) Create two sub-modules, *Generate* and *Monitor*, connected by a channel *ch1*. Create two variants of the design where the sub-modules are connected by a `sc_fifo<string>` or a `sc_signal<char>`. You will need an output port and an input port on each sub-module. Instantiate them inside *Hello*. Move the loop into the *Generate* module, but have it write to the output port. Have the *Monitor* display values that show up on the input port.



Sources for the *Hello* example are available at
http://www.ece.utexas.edu/~gerstl/ee382m_f18/hw/hw1.zip

Hello.h

```
#ifndef Hello_h
#define Hello_h
#include <systemc>
SC_MODULE(Hello) {
  SC_CTOR(Hello);
  void end_of_elaboration(void);
  void Hello_thread(void);
  ~Hello(void);
};
#endif
```

main.h

```
#include "Hello.h"
#include <iostream>
using namespace std;
using namespace sc_core;
int sc_main(void) {
  Hello top_i("top_i");
  cout << "Starting" << endl;
  sc_start();
  cout << "Exiting" << endl;
  return 0;
}
```

Hello.cp

```
#include "Hello.h"
#include <iostream>
using namespace std;
using namespace sc_core;
void Hello::Hello(sc_module_name nm)
: sc_module(nm) {
  cout << "Constructing "
       << name() << endl;
  SC_HAS_PROCESS(Hello);
  SC_THREAD(Hello_thread);
}
void Hello::end_of_elaboration(void) {
  cout << "End of elaboration" <<
endl;
}
void Hello::Hello_thread(void) {
  cout << "Hello World!" << endl;
}
Hello::~Hello(void) {
  cout << "Destroy " << name() <<
endl;
}
```