# Cache Simulator Project Report

## Project Description

The objective of this project was to design and implement a cache simulator using C++ with the optional use of the Qt framework. It simulates the performance of a cache with different memory capacities, block capacities, and memory mappings. The program is given an input file containing the byte address of different read and write calls which it processes an output report. Each line of the output file represents contains the cache configuration, the hit rate, total bytes transferred from cache to memory and vice versa, and the number of comparisons made to identify a hit.

## Implementation

The cache simulator was coded using the Qt framework. The file “simulate.cpp” contains the main while “cache.h” and “cache.cpp” represent the cache class. The cache class is modeled to represent a single type of cache whose memory capacity, block size, and mapping are supplied as arguments. It has two public functions for reading and writing which take the memory location as an argument. The read and write functions return a cacheInfo object which is a struct defined in the class header file that contains two booleans and an int. The first boolean identifies whether the operation had written to memory, the second is used to identify if there was a hit, and the int is used to store the number of comparisons. When “simulate.cpp” is executed, it stores the input file into a QStringList and runs a triple nested for loop which iterate through all the possible cache configurations. A new cache object is created per configuration which then processes all read and write requests contained in the QStringList using a fourth for loop. Once the input is exhausted all the information necessary for the current cache object is calculated and written to a single line of the output file.

## Runtime Configurations

The code was written on Qt 5.0.2. Both the input and output filenames are specified within “simulate.cpp” on the 17th and 18th line respectively. These should be changed when using different input and output files. The program itself is executed from the Qt Creator IDE.

## Trace Graphs

Figure 1: Hit Rate vs Associativity

The above graph shows the change in hit rate as the configuration is changed from direct mapping to associative mapping. The hit rate is vastly improved when using an associative mapping as opposed to direct mapping.

Figure 2: Hit Rate vs Block Size

The above graph shows the change in hit rate in a 1k direct mapped cache as the block size is increased. There is initial improvement in the hit rate however the gain is reversed when the block size reaches 64 bytes.